Уральский федеральный университет
имени первого Президента России Б.Н.Ельцина
Институт радиоэлектроники и информационных технологий

В. П. БИТЮЦКИЙ
С. В. БИТЮЦКАЯ

МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ АВТОМАТИЗАЦИИ ПРОЕКТИРОВАНИЯ

Учебное пособие

Министерство образования и науки Российской Федерации

Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина

В. П. Битюцкий
С. В. Битюцкая

МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ АВТОМАТИЗАЦИИ ПРОЕКТИРОВАНИЯ

Учебное пособие

Рекомендовано методическим советом УрФУ
для студентов специальности
230101 — Вычислительные машины,
комплексы, системы и сети

Екатеринбург
Издательство Уральского университета
2015

УДК 658.512.011.56
ББК 65.050.2
Б66

Рецензенты:
кафедра информатики Уральского государственного горного университета (зав. кафедрой — канд. техн. наук, доц. *А. В. Дружинин*),
канд. техн. наук, доц. *Г. Б. Захарова*

Научный редактор — канд. техн. наук, доц. *И. О. Ситников*

Битюцкий, В. П.

Б66 Математическое обеспечение автоматизации проектирования : учебное пособие / В. П. Битюцкий, С. В. Битюцкая.— Екатеринбург : Изд-во Урал. ун-та, 2015. — 72 с.
ISBN 978-5-7996-1447-8

В пособии рассматриваются процесс проектирования и его формализация. Подробно обсуждаются математические модели, которые используются при описании как процесса проектирования, так и объектов проектирования на техническом, функционально-логическом и блочном этапах проектирования.

Библиогр.: 4 назв. Табл. 18. Рис. 31.

УДК 658.512.011.56
ББК 65.050.2

Учебное издание

Битюцкий Валерий Петрович,
Битюцкая Светлана Валерьевна

МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ АВТОМАТИЗАЦИИ ПРОЕКТИРОВАНИЯ

Подписано в печать 15.04.2015. Формат 60×84 1/16. Плоская печать.
Усл. печ. л. 4,2. Уч.-изд. л. 3.9. Тираж 50 экз. Заказ № 159.

Редакционно-издательский отдел ИПЦ УрФУ
620049, Екатеринбург, ул. С. Ковалевской, 5
Тел.: 8 (343) 375-48-25, 375-46-85, 374-19-41
E-mail: rio@urfu.ru

Отпечатано в Издательско-полиграфическом центре УрФУ
620075, Екатеринбург, ул. Тургенева, 4
Тел.: 8 (343) 350-56-64, 350-90-13
Факс: 8 (343) 358-93-06
E-mail: press-urfu@mail.ru

ISBN 978-5-7996-1447-8

1. ФОРМАЛИЗАЦИЯ ПРОЦЕССА ПРОЕКТИРОВАНИЯ

1.1. Определение проектирования

Проектирование — процесс создания для еще несуществующего объекта, процесса или алгоритма функционирования (объект проектирования) описания, достаточного для реализации этого объекта в заданной технологической базе.

Под процессом понимается последовательность шагов принятия решения во времени, в результате которых

- устраняются некорректности в исходном задании,
- описание объекта уточняется,
- детализируется,
- оптимизируется,
- преобразуется и пр.

Каждый шаг процесса называется проектной процедурой, процедура состоит из проектных операций. Проектные операции — элементарные шаги преобразования, они могут быть достаточно сложными, но с точки зрения проектирования рассматриваются как неделимые на более мелкие.

Из определения проектирования следует, что исходное описание может быть некорректным. Определить и исключить некорректности — задача проектировщиков. Естественно, что эта задача должна решаться совместно с постановщиками задачи (заказчиками) через процедуру обсуждений и согласований.

1.2. Место проектирования в жизненном цикле изделия

Изделие может появиться в мире, когда, во-первых, в нем возникает необходимость и, во-вторых, когда существует возможность его реализации.

Жизненный цикл изделия включает в себя следующие этапы:

1. Проектирование изделия;
2. Изготовление опытного образца;
3. Технологическая подготовка производства (ТПП);
4. Серийное производство;
5. Эксплуатация;
6. Ремонт, модернизация;
7. Утилизация.

Таким образом, проектирование является первым этапом в жизненном цикле объекта (изделия). Жизненный цикл кроме собственного использования изделия включает и другие этапы. При проектировании возникает необходимость учитывать все этапы жизненного цикла, проектировать с учетом возможности реализации, минимальности затрат, удобства при изготовлении, эксплуатации, ремонта и модернизации, и наконец, утилизации.

Для сложного изделия сегодня рассматриваются еще этапы, связанные с обучением пользователей, предпродажной подготовкой, техническим сопровождением изделия и др.

1.3. Особенности процесса проектирования

Особенности процесса проектирования сложных изделий [2; 3]:

- Как правило, нет математических моделей описания объекта проектирования и процесса проектирования.
- Результат — описание предмета, которого еще не существует.
- Широко используется опыт проектировщиков, аналоги предыдущих решений, опыт других проектировщиков.
- Процесс проектирования требует участия специалистов различных областей, т. е. предполагается коллективная деятельность.
- Из возможных решений необходимо выбирать такие, которые должны обеспечить оптимальность реализации: наилучшие характеристики, минимальные затраты на изготовление, максимальную надежность и др.
- Необходимость итераций.
- Объект проектирования, как правило, принадлежит системе и сам является системой. Поэтому проектирование надо рассматривать как внешнее проектирование и внутреннее проектирование.
- Задача с многими критериями. Критерии противоречивые.

· Имеем дело с описанием, текстами (понимаемыми широко: это и рисунки, и чертежи и др.). Исходное задание и результат проектирования — тексты. Значит, процедуру можно представить как трансляцию одного текста в другой. Как и при программной трансляции, здесь могут использоваться библиотеки решений, дополнительные данные и т. д.

· Задача не может быть правильно сформулирована до тех пор, пока она не будет четко понята, т. е. в какой-то мере решена.

Эта особенность называется «принципом Маккола». Процесс решения задачи можно представить следующим образом (рис. 1). Мы поставили некоторую задачу, получаем некоторое частное решение, но не полное, следовательно, чтобы получить полное решение, надо сформировать новую задачу, но уже с имеющимся частным решением, и так до тех пор, пока не получим полного либо устраивающего нас решения задачи.

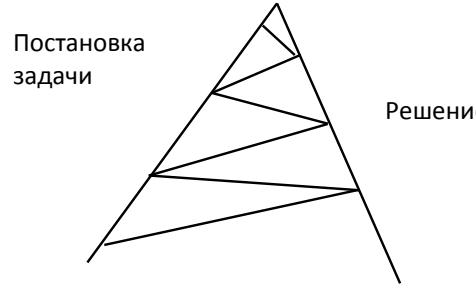


Рис. 1

1.4. Критерии проектирования

В простейших случаях в задаче с одним критерием существуют различные хорошо разработанные математические методы решения. Но как правило, процесс проектирования представляет собой решение многокритериальной задачи. В некоторых случаях удается свести задачу к одному критерию. Это бывает возможным, когда критерии имеют некоторую «однородность». Например, необходимо обеспечить экономичность функционирования объекта по различным затратам (электроэнергии, тепла, воды и др.). В этом случае вводят

цены c_i на каждый i-й вид затрат и минимизируют функционал стоимости функционирования

$$\sum_i c_i Q_i,$$

где Q_i — величина затрат i — го вида.

Такой подход дает возможность использовать реализованный метод проектирования для различных условий будущего функционирования объекта, например, проектировать объекты для функционирования в условиях более высоких цен на воду или на электроэнергию.

Когда критерии образуют два противоположных множества, типа «качество» и «затраты», и необходимо обеспечить максимальное значение качества при минимальных затратах, такие модели называют «двухполюсными». Для таких объектов используется прием сведения задачи к задаче с одним критерием. Один из критериев относят к ограничению, и при этом необходимо добиться оптимального значения второго критерия. Например, определяется, что стоимость объекта должна быть не более значения S max, и при этом необходимо обеспечить минимальную стоимость его производства.

Использование множества Парето. В общем случае критериев может быть много, они не сводятся к одному из рассмотренных выше случаев. Для сокращения области поиска решения выделяют множество возможных решений, которое называют множеством Парето. Это множество содержит только такие решения, для которых любая попытка улучшить значение по какому-то критерию требует ухудшения значения другого критерия. Разработаны методы выделения множества Парето, а затем из этого множества проектировщик выбирает решение по другим правилам. Здесь процедура выделения может быть сложной, и выделенное множество может быть слишком большим.

Поиск по дереву решений. Рассматриваются стратегии поиска: метод ветвлений, метод ветвей и границ, метод случайного поиска.

1.5. Блочно-иерархический подход в проектировании (БИП)

Разделение описаний проектируемых объектов на иерархические уровни по степени подробности отражения свойств объектов составляет сущность блочно-иерархического подхода к проектированию [1]. Соответственно, возможно разделение проектирования как

процесса на группы проектных процедур, связанных с получением и преобразованием описаний выделенных уровней. Эти группы процедур называются иерархическими уровнями проектирования.

Типичные иерархические уровни функционального проектирования БИС и СБИС: функционально-логический (на котором проектируются функциональные и логические схемы); схемотехнический (на котором разрабатываются принципиальные электрические схемы функциональных узлов и ячеек); компонентный (на котором решаются задачи проектирования элементов интегральных схем).

К типичным иерархическим уровням функционального проектирования ЭВМ относятся системный и функционально-логический уровни.

Иерархические уровни конструкторского проектирования ЭВМ связаны с разработкой конструктивов: стоек (шкафов), рам и панелей и др.

Иерархические уровни технологического проектирования выделяют в соответствии с группами задач проектирования принципиальных схем технологических процессов, маршрутной и операционной технологии.

При решении сложной задачи проектирования блочно-иерархический подход предполагает разбиение задачи на блоки с указанием структуры связей между этими блоками. Для каждого блока снова проводится разбиение на блоки (подблоки) и связи между ними, и так далее, до тех пор, пока блоки не становятся настолько простыми, что решение по ним принимается сразу, за один шаг (рис. 2).

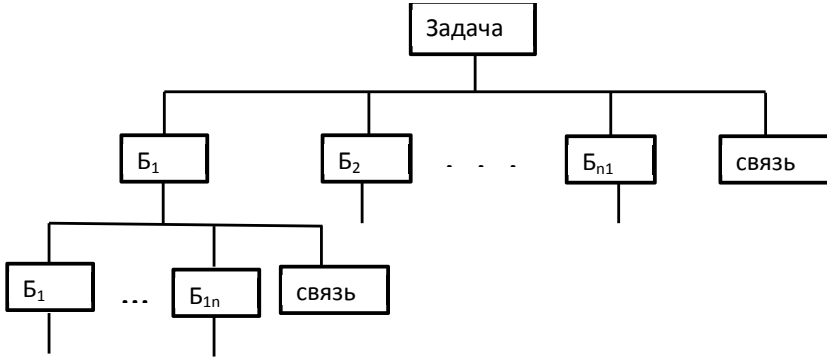


Рис. 2

В результате такого представления процесс проектирования разбивается на иерархические уровни — уровни абстрагирования.

Во многих случаях разбиение на блоки определяется семантикой задачи. Например, при проектировании завода выделяют части: строительную, сантехническую, электротехническую, КИП и автоматики, водоснабжение и др. При проектировании управляющей системы можно выделить техническую и программную части, в технической части — память, процессор, периферийную часть и пр. Число блоков должно быть небольшим, чтобы можно было видеть их взаимосвязь между собой.

На каждом уровне блочно-иерархического подхода решается задача минимизации связей между блоками.

1.6. Анализ и синтез в проектировании

При проектировании имеют дело с двумя типами описаний — функциональным (касается функционирования изделия) и структурным (описывает структуру из элементов нижнего уровня иерархии).

В соответствии с БИП, на каждом этапе решается задача построения по функциональному описанию структурного описывает и проверки структуры на соответствие функциональному описанию. Первая задача называется задачей синтеза, вторая — задачей анализа. Если анализ подтверждает соответствие описаний, переходят к следующему этапу, иначе — повторяют синтез структуры.

При решении задачи синтеза решаются две задачи: определение структуры и определение параметров входящих в структуру элементов. Рассматриваются две частных задачи. В первом случае параметры элементов заданы, и необходимо определить структуру из этих элементов. Эта задача называется задачей структурного синтеза. Такую задачу решают, когда необходимо найти схему, реализующую заданное множество функций или заданный абстрактный автомат. Предполагаются заданными элементы базиса со своими характеристиками—параметрами: время задержки, входное сопротивление и др.

Во втором случае задана структура, и необходимо найти параметры входящих в нее элементов в соответствии с параметрами структуры. Эта задача называется задачей параметрического синтеза.

Примером такой задачи может быть проектирование триггера с заданными характеристиками. Схема триггера определена, необходимо подобрать характеристики составляющего его элементов величину емкости для конденсаторов, сопротивления для резисторов и др.

2. ЭТАПЫ ПРОЕКТИРОВАНИЯ УПРАВЛЯЮЩИХ И ВЫЧИСЛИТЕЛЬНЫХ УСТРОЙСТВ

2.1. Технологический этап проектирования

На данном этапе рассматриваются возможности и способы изготовления объекта в данных условиях. Технологическая подготовка производства — это разработка наиболее экономичного процесса изготовления изделия, полностью отвечающего техническим требованиям. Исходные данные для технологической подготовки производства: конструкторская документация на проектируемое изделие, нормативно-техническая информация, данные о технологическом оборудовании. В процессе технологической подготовки производства решаются задачи: обеспечение технологичности конструкции изделия; проектирование оптимальных технологических процессов изготовления изделия и специальной технологической оснастки (фотошаблонов большие интегральные схемы и печатных плат, штампов, форм для отливок, приспособлений для сверления отверстий в печатных платах и т. п.); подготовка программ для программно-управляемого технологического оборудования, роботов манипуляторов, станков с числовым программным управлением (технологических автоматов). Технологическая документация (маршрутные и операционные технологические карты, эскизы технологических процессов, программы для технологических автоматов и т. п.), формируемая в процессе технологической подготовки производства, используется в качестве исходной информации в автоматизированных системах управления технологическими процессами и производством при изготовлении изделия.

2.2. Конструкторский (технический) этап проектирования

На этом этапе заданную логическую схему необходимо перевести в схему, представленную в конструктивах. Определены требова-

ния к реализации этой схемы на элементах заданной серии, Базовом матричном кристалле, БИС.

На этапе все задачи делятся на три группы: синтез конструкций, контроль полученных решений и выпуск документации.

Основная задача синтеза на конструкторском этапе — монтажно-коммутационное проектирование. Среди них различают задачи двух типов — задачи метрические и задачи топологические.

В метрических задачах ищутся и оцениваются метрические характеристики: размеры элементов, расстояние между ними, длина соединений, т. е. то, что можно оценить в миллиметрах, сантиметрах и т. п.

Топологические задачи определяют относительное расположение элементов и соединений в монтажно-коммутационном пространстве. Определяются такими характеристиками, как число пересечений связей, число межслойных переходов, число слоев печатной платы и т. п.

Основные метрические задачи:

- компоновка элементов нижнего уровня по элементам верхнего уровня;
- размещение элементов.

Основная топологическая задача — трассировка соединений элементов на каждом поле.

2.2.1. Компоновка

Задача компоновки решается в двух вариантах.

1. Компоновка функционально типизированными узлами: в этом случае схему необходимо разбить на блоки, взятыми из заданного множества. Эта задача называется еще задачей покрытия.

Пример. Задана схема, построенная в конъюнктивно-дизъюнктивном базисе. Необходимо «вложить» ее в элементы заданной серии по корпусам. Каждый корпус характеризуется составом элементов, с указанием числа их входов, связями их в корпусе.

2. Компоновка без функциональной типизации. В этом случае заданы ограничения на блок (в виде, например, числа корпусов в блоке, числа внешних выводов блока).

Пример. Заданную схему в корпусах необходимо разместить по платам заданных размеров.

Во втором варианте возможны классы задач.

Если S — число соединений между блоками, k_i — число элементов в i-м блоке, q_i — *число* выводов в i-м блоке, то:

- Задача 1. Получить min S при $k_i \le k_0$, $q_i \le q_0$.
- Задача 2. Получить min max q_i при $k_i \le k_0$ (равномерность распределения выходов по блокам).
- Задача 3. Получить минимальное число блоков при $k_i \le k_0$, $q_i \le q_0$.

Имеется пять методов (групп алгоритмов) решения задачи компоновки:

1. Последовательные алгоритмы;
2. Итерационные;
3. Смешанные;
4. На основе целочисленного программирования;
5. На основе метода ветвей границ.

Методы 4 и 5 являются точными, требуют большого перебора, поэтому используются редко.

Последовательные методы

Идея алгоритма состоит в последовательной укладке элементов по блокам.

Множество элементов разбивается на два подмножества:

- назначенные в блок элементы,
- не назначенные в блок элементы.

Последовательно укладываем элементы в блоки, так, чтобы для каждого элемента выполнялось условие:

- он должен быть максимально связан с элементами блока (p_1 связей);
- он должен быть минимально связан с остальными элементами (p_2 связей);
- разность $p_1 - p_2$ должна быть максимальной.

Итерационные методы

Элементы каким-то образом уже распределены по блокам. Меняем элементы из блоков между собой, оценивая при этом изменение стоимости нового распределения.

Смешанные методы

С помощью последовательного алгоритма формируется начальное размещение элементов по блокам, затем применяется итерационный алгоритм для оптимизации полученной компоновки.

Пример

Рассмотрим порядок компоновки следующей схемы (рис. 3) в два блока при помощи последовательного алгоритма:

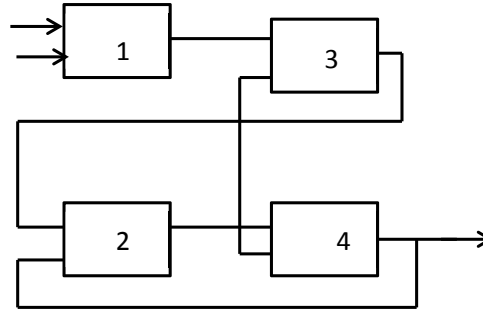


Рис. 3

1) Элемент 1 кладем в блок 1.

2) Элемент 2 не имеет связей ни с блоком 1, ни с блоком 2, положим его в блок 2.

3) Элемент 3: если его положить в блок 1, то он будет иметь одну связь с этим блоком и две связи с блоком 2, т. е. $p_1 - p_2 = 1 - 2 = -1$. Если же положить его в блок 2, то будет две связи с этим блоком и одна связь с остальными, т. е. $p_1 - p_2 = 1$. Таким образом, элемент 3 нужно положить в блок 2.

4) Для элемента 4 осталось только одно место — в блоке 1.

Как видим, слабым местом последовательного алгоритма является недостаток информации о связях между блоками в начале компоновки. К концу компоновки, когда эта информация становится полной, последние элементы уже приходится расставлять по оставшимся свободным местам, которые далеко не всегда бывают оптимальными.

Можно попытаться устранить этот недостаток с помощью итерационного алгоритма, применив его к полученной компоновке. В нашем примере имеет смысл поменять местами элементы 3 и 4. Было:

элемент 1: ноль связей с блоком 1, одна связь с остальными;

элемент 4: ноль связей с блоком 1, три связи с остальными;

элемент 2: две связи с блоком 2, три связи с остальными;

элемент 3: две связи с блоком 2, одна связь с остальными.

После перестановки 3 и 4:

элемент 1: одна связь с блоком 1, ноль связей с остальными;

элемент 3: одна связь с блоком 1, две связи с остальными;

элемент 2: три связи с блоком 2, две связи с остальными;

элемент 4: три связи с блоком 2, ноль связей с остальными.

После перестановки увеличилось количество связей внутри каждого блока и уменьшилось количество связей между блоками.

2.2.2. Размещение

Задача размещения заключается в том, чтобы на основе принципиальной схемы устройства и информации об упаковке элементов по блокам разместить эти элементы в блоке (например, на печатной плате) наилучшим образом.

Элементы могут размещаться как на одной, так и на обеих сторонах печатной платы. Размещение может быть оптимизировано по таким параметрам, как

- плотность заполнения платы,
- суммарная длина связей между элементами и др.

Алгоритмы размещения компонентов на плате в целом аналогичны алгоритмам компоновки. Точно так же основными алгоритмами размещения являются последовательный алгоритм и итерационный алгоритм.

Последовательный алгоритм

Сначала размещаются элементы, для которых положение на плате известно заранее (например, разъемы), или элементы с максимальным количеством связей с другими. Далее, элементы последовательно размещаются на плате таким образом, чтобы суммарная длина связей с уже размещенными элементами была максимальной. Метод обладает тем же недостатком, что и последовательный метод компоновки.

Итерационный алгоритм

Итерационный алгоритм помогает оптимизировать начальное размещение элементов путем перестановки некоторых элементов и оценки полученного результата. Мы использовали такую модель: пусть как-то элементы размещены. Зафиксируем первый и третий уровни (первый уровень — это фиксированный разъем). Представим, что элементы второго уровня располагаются на условном «стержне», а связям соответствуют натянутые резинки или пружинки. Отпустим элементы второго уровня, под действием пружинок они займут положения, которые будут соответствовать минимуму натяжений. За-

фиксируем это положение и перейдем к определению расположения на третьем уровне, закрепив еще и элементы четвертого уровня. Для такой модели составлены соответствующие математические уравнения, решая которые, получим некоторое локальное распределение элементов.

2.2.3. Трассировка

Задачей трассировки является прокладка на печатной плате соединений между элементами. При трассировке учитывается стоимость прокладки проводников по каждому направлению, стоимость сверления переходных отверстий, минимизируется число изгибов проводников, а также ряд других параметров.

Алгоритмы трассировки:

- волновой,
- лучевой,
- канальный,
- магистральный.

Волновой алгоритм Ли

Волновой алгоритм Ли имитирует распространение сигнала в среде. Источником сигнала является одна из точек, между которыми прокладывается соединение. Все соседние с ней точки также становятся источниками волны следующего уровня. И так продолжается до тех пор, пока волна не дойдет до нужной точки. После этого точки соединяются в обратном порядке:

16 – 15 – 14 — ...— 0.

Ускорить процесс можно, если пустить волну одновременно из обеих точек навстречу друг другу.

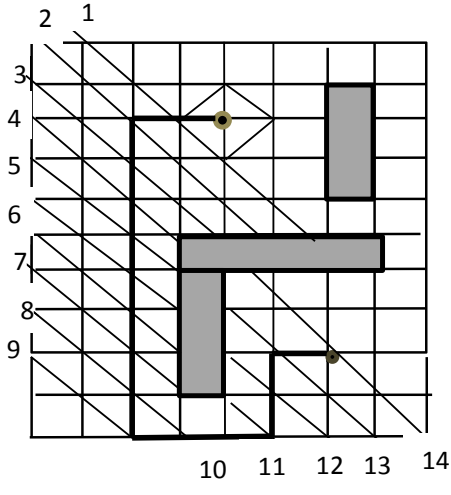


Рис. 4

На рис. 3 показан результат трассировки по алгоритму Ли. Здесь затемненными пря-

моугольниками выделены зоны запрета, где трассы проводить нельзя, цифрами указаны шаги выполнения алгоритма.

Магистральный алгоритм

В магистральном алгоритме трассировки соединение строится следующим образом: начинаем движение из одной точки и делаем шаг в некотором направлении (горизонтальном или вертикальном), пытаясь приблизиться к нужной точке. Если на пути встречается препятствие, возвращаемся на шаг и меняем направление.

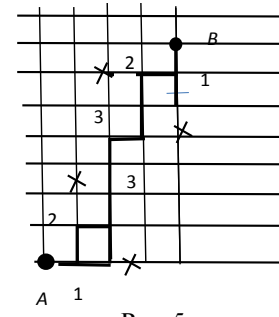


Рис. 5

Можно также двигаться навстречу из обеих точек до пересечения магистралей.

Канальный алгоритм

Канальный алгоритм трассировки предполагает размещение всех связей между элементами в канале связи. Задача трассировки сводится к укладке отрезков по каналу.

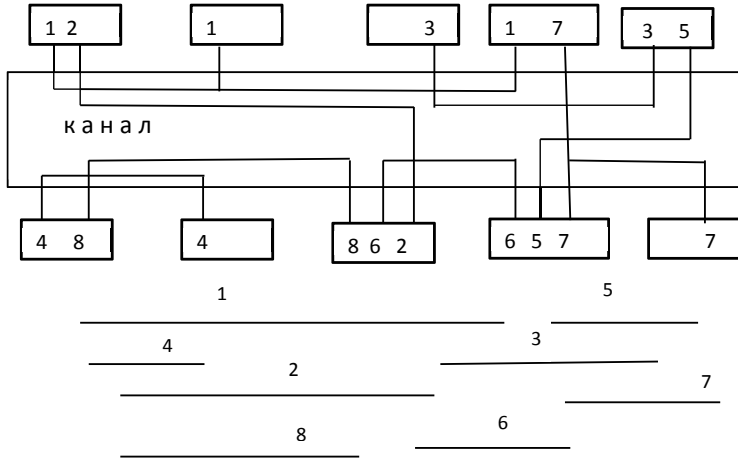


Рис. 6

3. МАТЕМАТИЧЕСКИЕ МОДЕЛИ ТЕХНИЧЕСКОГО ЭТАПА

3.1. Бинарные отношения

Пусть заданы два множества A и B. Выделим некоторое подмножество $\Re$ декартова произведения $A \times B$ и будем трактовать его элементы (a_i, b_j) как выражение того факта, что a_i и b_j находятся в некотором соответствии. Нас интересует не характер этого соответствия, а только сам факт его наличия. Множество $\Re$ назовем отображением множества A на множество B.

Если $(a_i, b_j) \in \Re$, то a_i называется прообразом b_j, а b_j — образом a_i при отображении $\Re$. Множество $A^{\sim}$ всех прообразов в A есть область определения отображения $\Re$, а множество $B^{\sim}$ всех образов в B — область значений $\Re$. Если $B^{\sim}$ равно B, то говорят об отображении на B, если $B^{\sim}$ — только часть B, то об отображении в B.

Отображение будем обозначать как $A \Re B$ или $\Re(a) = b$. Если для каждого a_i образ b_j единственен, то отображение называют функциональным. Если в $\Re$ все пары (a_i, b_j) переписать «наоборот», как (b_j, a_i), получим отображение B в A, которое является обратным к $\Re$ и обозначается $\Re^{-1}$.

Пусть множества A и B совпадают, $\Re \subset A \times A$. В этом случае $\Re$ называют бинарным отношением, а множество A — базовым множеством отношения $\Re$.

Если $(a_i, a_j) \in \Re$, то говорят, что элемент a_i находится в отношении $\Re$ с элементом a_j. В общем случае можно определить, что в отношении находится не пара, а k элементов, считать, что $\Re \subseteq A^k$. Величина k определяет арность отношения $\Re$. Говорят, что $\Re \subseteq A^k$ — k-арное отношение.

Термин «отношение» используют также, если арность > 2 и множества в декартовом произведении различны.

Будем рассматривать в дальнейшем бинарные отношения на множестве A.

3.1.1. Способы описания бинарного отношения

Бинарное отношение $\Re$ как любое подмножество может быть представлено в виде перечисления, через указания свойства или через порождающую процедуру. Наиболее часто используется представление *матрицей*. Бинарное отношение называется *рефлексивным*, если $\forall(a_i) \in A \ \ (a_i, a_j) \in \Re$. Если отношение рефлексивное, то в каждой клетке главной диагонали стоят единицы.

Бинарное отношение *антирефлексивно*, если $\forall(a_i) \in A$ $(a_i, a_j) \notin \Re$. В антирефлексивном отношении главная диагональ не содержит ни одной единицы.

Бинарное отношение называется *симметричным*, если из того, что $(a_i, a_j) \in \Re$, следует $(a_j, a_i) \in \Re$. Для симметричного отношения таблица симметрична относительно главной диагонали.

Бинарное отношение *антисимметрично*, если из того, что $(a_i, a_j) \in \Re$, следует, что $(a_j, a_i) \notin \Re$.

Бинарное отношение называется *транзитивным*, если из того, что $(a_i, a_j) \in \Re$ и $(a_j, a_k) \in \Re$, следует $(a_i, a_k) \in \Re$.

Бинарное отношение называется *отношением эквивалентности*, если оно одновременно рефлексивно, симметрично и транзитивно.

Два элемента связаны отношением эквивалентности, если они имеют одинаковое свойство из множества альтернативных свойств. Примерами таких отношений являются принадлежность студентов к одной учебной студенческой группе, отношение родства или отношение «иметь одинаковый цвет волос». Альтернативность предполагает, что случаи, когда один студент принадлежит к нескольким группам или один человек имеет разноцветные волосы, из рассмотрения исключаются (иначе не выполнялась бы транзитивность). Тогда множество разбивается на непересекающиеся подмножества элементов, удовлетворяющие свойству, которые при объединении покрывают все множество. Последнее обеспечивается свойством рефлексивности, когда для каждого элемента находится элемент, с которым он состоит в отношении (по крайней мере, с самим собой). Эти подмножества называются классами эквивалентности.

Справедливо утверждение: *любому отношению эквивалентности однозначно сопоставляется разбиение множества, и обратно, любому разбиению множества однозначно сопоставляется отношение эквивалентности.*

3.2. Графы

Граф является одним из способов описания бинарного отношения, рассмотренного в предыдущем разделе. Легко видеть, что бинарное отношение на множестве — наиболее общее описание связей между элементами множества, указывающее факт наличия или отсутствия связи между парами элементов из множества независимо от характера связи.

Определение. Графом G называют пару $<A, U>$, где

$A = \{a_1, a_2, ..., a_n\}$ *–множество вершин графа;*

$U \subseteq A \times A = \{(a_i, a_j)\}$ *— множество его дуг.*

Последовательность дуг, в которой конец каждой предыдущей дуги совпадает с началом последующей дуги, называют путем между вершиной — началом первой дуги (началом пути) и вершиной — концом последней дуги (концом пути). Число дуг в последовательности принимается за длину пути.

3.2.1. Поиск путей в графе

Пусть в графе заданы две вершины $a_н$ и $a_к$, названные соответственно начальной и конечной. Выделим несколько задач, связанных с поиском путей в графе:

- найти путь между начальной и конечной вершиной. Эта задача называется задачей поиска пути в лабиринте;
- найти минимальный путь между заданными вершинами;
- найти максимальный путь;
- найти цикл Эйлера.

Рассмотрим задачу поиска минимального пути между двумя заданными вершинами $a_н$ и $a_к$ в графе при условии, что каждой дуге (i, j) сопоставлен вес $c_{i,j}$ – неотрицательное число и оценка аддитивна. Если веса обозначают длину дуги, то задача формулируется как задача нахождения кратчайшего пути между заданными вершинами.

Рассмотрим классический алгоритм решения этой задачи — алгоритм Дейкстры, в основе которого лежит следующий *тезис Дейкстры*: если кратчайший путь проходит через вершину a_i, то длина части пути от $a_н$ до a_i должна быть минимально возможной.

Алгоритм представим следующей последовательностью шагов:

1. Начальные присваивания. Каждой вершине, кроме начальной, сопоставим вес $l(a_i)$, равный бесконечности, назовем этот вес

временным. Начальной вершине сопоставим вес, равный нулю: $l(a_н) = 0$, назовем этот вес постоянным, вершину $a_н$ назовем текущей и обозначим как P.

2. Обновление весов. Всем вершинам, связанным с текущей по исходящим дугам и имеющим временные веса, изменим веса по правилу $l(a_i) = \min(l(a_i), l(P) + c_{P,i})$.

3. Смена текущей вершины. Среди вершин с временной оценкой найдем вершину с минимальным весом и назовем ее текущей, а ее вес — постоянным. Если это есть вершина $a_к$, то перейдем к пункту 4, иначе — к пункту 2.

4. Выделение пути обратным ходом. Определим конечную вершину как текущую; для каждой вершины, связанной с ней по заходящим дугам, определим разность между весом текущей вершины и весом дуги. Вершину, вес которой совпадает с этой разностью, назовем текущей, а дугу отнесем к пути. То, что такая вершина всегда найдется, гарантируется способом построения весов вершин. Повторим эту процедуру до тех пор, пока текущей не станет начальная вершина. В результате множество отнесенных к пути дуг даст искомое решение.

Путь максимальной длины есть смысл искать только в графах, где нет контуров. Действительно, в графе с контурами решение однозначно и равно бесконечности, ибо существует путь, бесконечное число раз проходящий по этому контуру. Одной из задач, где используется поиск максимального пути, является задача анализа сетевого графика. Дадим необходимые определения.

Семантика задачи. Сопоставим каждой дуге работу. Вес дуги — время выполнения работы. Вершине сопоставим событие, состоящее в том, что все работы, приписанные заходящим в нее дугам, выполнены, и можно начинать все работы, приписанные исходящим дугам. Граф с такой трактовкой называется *диаграммой ПЕРТ* (от *Program EvalUation and Review TechniqUe*) или *сетевым графиком.*

В сетевом графике максимальному пути, называемому критическим, будет соответствовать минимальное время, необходимое для выполнения всех работ. Для сокращения общего времени можно рассматривать, например, вопрос об автоматизации работ на критическом пути с целью сокращения времени их выполнения.

Для решения задачи поиска максимального пути можно воспользоваться алгоритмом Дейкстры, изменив его соответствующим образом. Однако задано добавочное условие на граф задачи: в нем нет

контуров. Добавочные условия могут потребовать проверки их выполнения и усложнить алгоритм, но можно использовать эти условия и для сокращения алгоритма, что мы и сделаем.

Справедлива следующая теорема: *если в графе нет контуров, то его вершины можно перенумеровать таким образом, что всякая дуга идет из вершины с меньшим номером в вершину с большим номером,* и при этом возможна последовательная нумерация вершин от 1 до $n = |A|$.

Будем использовать широко распространенный в программировании прием для сокращения решения комбинаторных задач: сначала тратятся добавочные усилия на предварительное упорядочение, затем в упорядоченном множестве более просто находят решение.

1. Упорядочим множество вершин так, чтобы любая дуга исходила из вершины с меньшим номером и заходила в вершину с большим номером. Это можно сделать так: присвоим начальной вершине номер 1. Всем вершинам A', связанным с начальной по исходящим дугам, сопоставим номера от 2 до $|A'|$. После этого проверим номера этих вершин на выполнение условия. Для любой пары вершин, где условие не выполняется, переставим номера вершин. Затем рассматриваются вершины, связанные по исходящим дугам с множеством A', и т. д.

2. Присвоим всем вершинам вес $l(i) = 0$.

3. Последовательно для вершин 1, 2, 3, ... проведем пересчет весов по формуле $l(i) = \max(l(i), l(j) + C_{ji})$, где максимум берется по всем вершинам j, из которых есть дуги в вершину i. Это можно сделать, так как ко времени пересчета вершины i веса всех вершин j вычислены раньше, ибо $i > j$. Сокращение вычислительных затрат по сравнению с алгоритмом Дейкстры связано с тем, что отпадает необходимость на каждом шаге определять очередную вершину с минимальным весом для продолжения расчетов.

4. Как и в алгоритме Дейкстры, выделим обратным ходом искомый путь.

3.2.2. Деревья

Определение: неориентированным деревом называют граф, удовлетворяющий одному из условий:

- связный граф без циклов;
- связный граф из n вершин и $(n - 1)$ ребра;
- граф, в котором любые две вершины связаны ровно одной цепью.

Задача: докажите, что все три условия описывают один и тот же объект и что из того, что граф обладает одним из этих свойств, следуют два других.

Ориентированным деревом называют связный граф, в котором в каждую вершину, кроме одной, называемой корнем дерева, заходит ровно одна дуга. В корень дерева ни одна дуга не заходит.

Вершины, из которых не выходит ни одна дуга, называются листьями.

Деревья используются для описания структур организаций, предприятий и др. Такие структуры называются иерархическими. Примером может быть структура управления, где корень дерева — управляющий, с ним связаны непосредственно подчиненные ему руководители — вершины 1-го уровня, которым, в свою очередь, непосредственно подчинены другие — вершины 2-го уровня, и так вплоть до исполнителей нижнего уровня — листьев. Дерево образует структура предприятия, где корень — само предприятие, под ним — входящие в него цехи и службы, ниже — входящие в цехи участки и т. д. Принято дерево изображать корнем вверх.

3.2.3. Дерево решений

В жизни мы постоянно сталкиваемся с проблемой принятия одного решения из множества возможных — с проблемой выбора. В результате выбора мы попадаем в новую ситуацию, когда снова нужно делать выбор.

Возможности выбора при решении проблемы можно представить в виде дерева, где в корне — проблема, дуге соответствует один из вариантов выбора, вершине — новая ситуация, возникающая в результате реализации приписанного дуге варианта. Такой трактовке соответствует граф типа дерева, получивший название «дерево решений». Предположим, что можно оценить эффективность принятого выбора. Тогда возникает задача поиска среди возможных путей от корня (когда проблема поставлена) к одному из листьев (когда проблема решена) пути, имеющего оптимальную оценку.

Стратегии поиска по дереву решений. Предположим, что дерево решений очень велико, поэтому поиск путем полного перебора всех возможных решений невозможен. Рассмотрим стратегии поиска, связанные с сокращенным перебором решений: метод ветвлений и метод ветвей и границ.

Метод ветвлений. Поиск начинается от корня дерева. Выбирают минимальную по стоимости исходящую дугу и по ней переходят к следующей вершине. Если эта вершина не является листом, снова переходят по минимальной исходящей дуге, пока не будет достигнут лист дерева. Алгоритм очень просто реализуется, но очевидно, что при этом решение может быть далеким от минимального.

Метод ветвей и границ. На каждом шаге выделяется множество вершин в качестве границы, веса вершин которой оцениваются длиной пути до нее от корня. На первом шаге граница состоит из единственной вершины — корня дерева, на втором шаге границу образуют все вершины первого уровня, которым приписываются веса, равные сумме веса корня, принимаемого равным нулю, и веса соответствующей дуги от корня до вершины. На границе находится вершина с минимальной оценкой. Если эта вершина является листом, то решение найдено — путь до этого листа и будет минимальным решением, если нет, то строится новая граница заменой найденной вершины на вершины, связанные с ней по исходящим дугам. Этим вершинам приписываются веса, равные сумме веса заменяемой вершины и веса дуги. Нетрудно понять, что этот вес определяет длину ее пути от корня. Поиск решения продолжается до тех пор, пока вершина с минимальным весом на границе не окажется листом. Метод гарантирует получение минимального решения, однако часто связан с большим объемом вычислений.

3.2.4. Поиск минимального остова

Остовом графа G называется его частичный граф типа дерево. Таким образом, остов выделяется из графа G удалением из него ребер до тех пор, пока он не станет графом типа дерево, включающим все вершины.

Если ребра графа взвешены, то возникает задача выделения остова с минимальной или максимальной оценкой.

Семантика задачи. Предположим, что вершинам графа сопоставлены полюса схемы, на которые необходимо подать питание, а ребрам — разрешенные связи для цепи питания. Тогда любой из остовов будет определять вариант цепи питания. Действительно, цепи сопоставляется граф без петель, включающий все вершины. Если веса определяют расстояние между полюсами, то остову с ми-

нимальной суммой весов соответствует разводка питания с минимальной суммарной длиной проводников. Такая задача решается при трассировке печатных плат.

Может быть предложена такая трактовка задачи. Вершины — абоненты, соединенные линиями связи, веса на ребрах — оценка потери конфиденциальной информации при ее передаче по этой линии связи. Тогда задаче выделения остова с минимальным произведением весов, входящих в остов ребер, соответствует задача определения наиболее надежной сети для передачи информации.

Рассмотрим два алгоритма решения задачи: жадный алгоритм и алгоритм Прима.

Жадный алгоритм

Выбирается ребро, имеющее минимальный вес среди всех ребер, и включается в остов. Из оставшихся ребер выбирается снова то, которое имеет минимальный вес, и включается в остов, если при этом не образуются циклы. Процесс продолжается до тех пор, пока все вершины не будут включены в остов.

Алгоритм прост для понимания и обеспечивает получение минимального решения. Однако сложность его состоит в том, что в нем неявно присутствует процедура проверки на появление циклов, которая связана с перебором по всему графу, так же как и поиск очередного ребра.

Алгоритм Прима

1. Включим любую вершину в остов.

2. Рассмотрим все ребра, исходящие из вершин, включенных в остов к оставшимся вершинам, и из них выберем ребро с минимальным весом. Его концевую вершину включим в остов. Повторяем этот пункт, пока не все вершины включены в остов.

В этом алгоритме не нужно следить за образованием циклов. Алгоритм начинает работу с произвольно выбранной вершины. Кроме того, на каждом шаге просматривается только одна строка матрицы смежности.

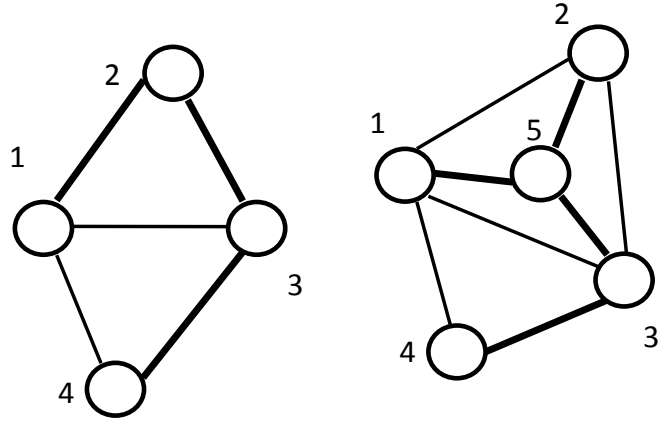
3.2.5. Деревья Штейнера

Пусть на двумерной плоскости расположены вершины графа, связанные ребрами. Ребро взвесим расстоянием между вершинами, связанными этим ребром. Допустим возможность вводить произ-

вольным образом добавочные вершины, называемые точками Штейнера, и заменять ребра цепью из ребер, проходящей через точки Штейнера. Задачу выделения остова в таком расширенном графе называют задачей Штейнера, а выделенный остов — деревом Штейнера (по фамилии математика, занимающегося этой задачей).

Задача Штейнера имеет очевидную инженерную интерпретацию: вершинам сопоставляются эквипотенциальные полюса сети, например полюса, на которые должно быть подано питание. Ребрам соответствуют допустимые способы связи между полюсами. Тогда решению будет соответствовать электрическая цепь минимальной длины, объединяющая все эти вершины. Точки Штейнера — точки «разветвления» в цепи, известны в инженерной практике как «Т-цепи». В зависимости от метрики пространства, в котором рассматривается задача, возможны следующие задачи Штейнера.

Пусть ребром связаны вершина 1 с координатами (x_1, y_1) и вершина 2 с координатами (x_2, y_2). Если расстояние между вершинами 1 и 2 (длина дуги (1,2)) определяется, как $L\ (1,2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$, то задача называется евклидовой (рис. 8).



а *б*

Рис. 8

Если расстояние определяется как $L(1,2) = |x_1 - x_2| + |y_1 - y_2|$, то задача называется линейной (рис. 9). С линейной метрикой имеют дело при работе с печатными платами, где связь возможна только в решетке по взаимно перпендикулярным направлениям.

Для евклидовой задачи установлено: вершины Штейнера имеют степень три и располагаются в таком месте, что углы между инцидентными ребрами равны 120°.

Если пространство линейное, то в качестве мест для расположения точек Штейнера достаточно рассмотреть только точки пересечения вертикальных и горизонтальных линий, проходящих через вершины графа. Это резко сокращает число возможных вариантов. В примере для линейного случая (рис. 9) точка Штейнера — вершина 5.

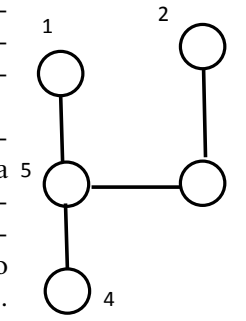


Рис. 9

3.2.6. Паросочетания

Определение. *Паросочетанием* в графе называется такое подмножество его ребер, в котором никакая пара ребер не смежна.

Интерес представляют две задачи, связанные с нахождением паросочетаний. Если паросочетание оценивать числом входящих в него ребер, то наибольшим называют паросочетание, имеющее наибольшее число ребер.

Если ребрам приписаны веса, то максимальным называют паросочетание, для которого сумма весов входящих в него ребер максимальна.

Алгоритм нахождения наибольшего паросочетания использует следующие понятия.

Вершину в графе назовем связанной, если она принадлежит паросочетанию, иначе вершину назовем свободной.

Цепь назовем последовательной, если в ней последовательно повторяются ребра, принадлежащие и не принадлежащие паросочетанию.

Если все ребра паросочетания можно включить в последовательную цепь, в которой начальная и конечная вершины свободные, то найдется паросочетание, в котором будет число ребер на единицу больше. Оно получается, если в выделенной цепи все ребра, не входящие в паросочетание, включить в него, а все входящие — исключить.

Алгоритм основан на этом утверждении. Строится последовательная цепь и описанной процедурой увеличивается число ребер в паросочетании, пока это возможно.

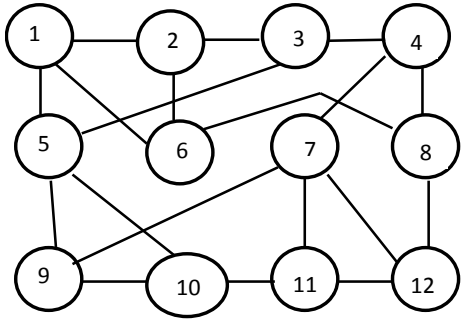


Рис. 10

Пример. Рассмотрим граф, приведенный на рис. 10. Выделим в нем последовательную цепь, например 1, 2, 3, 5, 10, 11, 7, 12, 8, 4, в которой ребра (1,2), (3,5), (10,11), (7,12), (8,4) включены в паросочетание.

Для этой цепи ищем цепь, в которую пробуем включить свободные вершины 6 и 9 так, чтобы они были начальной и конечной вершинами цепи. Такая цепь находится (9, 10, 11, 7, 12, 8, 4, 3, 5, 1, 2, 6), значит, получаем новое паросочетание, в котором число ребер на единицу больше первого. Так как это паросочетание содержит все вершины, оно является наибольшим. Решением будет множество {(9, 10), (11, 7), (12, 8), (4, 3), (5, 1), (2, 6)}.

Можно убедиться, что это решение не единственное.

Паросочетание называют совершенным, если в нем участвуют все вершины. Очевидно, что совершенное паросочетание, если оно в графе существует, будет и наибольшим.

Так как с каждым ребром связана пара вершин и ребра в паросочетании не смежные, то с любым паросочетанием связано четное число вершин. Значит, имеет место теорема.

Теорема. Необходимым условием существования в графе совершенного паросочетания является четное число его вершин.

3.2.7. Задача о назначениях

Задача нахождения максимального или наибольшего паросочетания в двудольном графе $G =< A \cup B, U >$, $U \supseteq A \times B$ называется задачей о назначениях.

Не теряя общности, можно предположить, что $|A| = |B|$. (Если это не так, то всегда можно меньшее множество дополнить элементами, не связанными с другими элементами).

Будем решать следующую задачу: найти в двудольном графе максимальное совершенное паросочетание.

В терминах работников и должностей множеству А сопоставляется множество работников, множеству В — множество должностей, ребра взвешены числами (целыми и положительными), определяющими эффективность назначения работника на должность. Число работников равно числу должностей. Нужно найти такое назначение, когда все работники назначены на должности, все должности заняты, и при этом достигается максимальная эффективность использования работников.

Условие существования совершенного паросочетания в двудольном графе определяется теоремой Кенига — Холла.

Теорема Кенига—Холла: совершенное паросочетание существует тогда и только тогда, когда для любого $A' \subseteq A$ имеет место $|A'| \leq |U(A')|$.

Метод выделения максимального паросочетания заключается в следующем (этот метод в литературе называют венгерским).

Взвесим вершины графа весами x_i для вершин множества А и y_iдля вершин множества В таким образом, чтобы условие $c_{ij} \leq x_i + y_i$ удовлетворялось для любого ребра. Если в решении оставлять только те ребра, для которых выполняется равенство $x_i + y_i = c_{ij}$ (*), и обеспечить подбором x_i и y_i выполнение условия теоремы Кенига — Холла, то при этом сумма c_{ij}в паросочетании (которое существует по теореме Кенига — Холла) будет максимально возможной, а сумма всех x_i и y_i — минимально возможной, так как первая сумма ограничивает вторую сверху, вторая сумма ограничивает первую снизу. В силу условия (*) при этом $\sum c_{ij} = \sum x_i + \sum y_j$.

Начинается расчет с того, что значению x_i приписывается $max\ c_{ij}$ по всем j, а y_i –нулевые значения. Выбираются в претенденты только те ребра, для которых выполняется равенство (*) и проверяется условие теоремы Кенига–Холла.

Если находится подмножество А», для которого теорема не выполняется, то для каждого его элемента вес x_i уменьшается на единицу, а вес каждого элемента из $U(A')$ уменьшается на единицу. Если появляются новые ребра, для которых выполняется условие (*), их вносят в претенденты и снова проверяются условия теоремы. Это продолжается, пока не будут выполнены условия теоремы.

В графе, содержащем только претенденты в решение, решение находится путем перебора. При этом вначале для всех вершин, степень которых равна 1, то есть для которых назначения однозначны,

производятся назначения, соответствующие пары вершин из графа претендентов удаляются, что сокращает объем перебора.

3.2.8. Цикломатическое число графа

Пусть в графе m — число ребер, n — число вершин, P — число компонент связности. Величина $r = n - P$ называется коцикломатическим числом. Цикломатическим числом графа называют число $n = m - n + P$.

Формально понятие независимых циклов вводится следующим образом. Введем в графе произвольную ориентацию ребер. Рассмотрим некоторый произвольный цикл и сопоставим ему вектор, который содержит m компонент, сопоставленных ребрам по правилу: значение компоненты $r_{ij} = r^{+}_{ij} - r^{-}_{ij}$, где r^{+}_{ij} — число проходов по циклу по направлению ребра, r^{-}_{ij} — число проходов против направления ребра.

Введем две операции на множестве всех векторов: сложение векторов как покомпонентное сложение их компонент и умножение вектора на число как умножение каждой компоненты на это число.

Множество векторов называется линейно независимым, если равенство $a_1 r_1 + a_2 r_2 ... + a_к r_к = 0$ выполняется только при $a_1 = a_2 = ... = a_к = 0$. Множество векторов с введенными операциями образует линейное пространство. Множество, содержащее максимальное число его линейно независимых векторов, называется базисом пространства, число элементов базиса называется размерностью пространства. В этом пространстве любой вектор может быть выражен как сумма произведений базисных векторов на константы.

Теорема. В графе число n определяет число независимых циклов в нем.

Доказательство проводится по индукции. Предположим, что теорема справедлива для числа ребер m, и покажем, что тогда она справедлива и для $m` = m + 1$. Добавим в граф G еще одно ребро (a, b), тогда возможны два случая:

1. В G вершины a и b связаны цепью, тогда в расширенном графе G'добавится еще один цикл, т. е. будет $m'» = m + 1$, $P' = P$, $n' = n + 1$.

2. В G вершины a и b цепью не связаны, тогда в расширенном графе G' уменьшится на единицу число компонент связности, т. е. $m' = m + 1$, $P' = P - 1$, $n' = n$.

Для любого графа проделаем такую процедуру: удалим из него все ребра и затем введем их по одному. Для графа без ребер утверждение теоремы верно (число циклов и число ребер равны 0, число вершин равно числу компонент связности). Каждое новое ребро будет приводить к описанным выше двум случаям, т. е. будет выполняться утверждение теоремы.

Если графу сопоставить электрическую сеть, то r — наибольшее число независимых разностей потенциалов в сети между ее узлами, n — число независимых круговых токов, которые могут протекать в этой сети.

3.2.9. Планарные графы

Плоскими графами называют графы, которые можно так расположить на плоскости, чтобы ребра пересекались только в вершинах графа. Графы, которые расположены на плоскости без пересечения, называют планарными. Планарные графы, получающиеся один из другого непрерывной деформацией плоскости, не считаются различными.

Часть плоскости, ограниченная ребрами и не содержащая внутри себя ни вершин, ни ребер, называется гранью. Одна из граней является внешней. Грани, разделенные ребрами, называются смежными.

Для планарного графа число граней f связано с числом его вершин n и числом его ребер m формулой Эйлера $n - m + f = 2$.

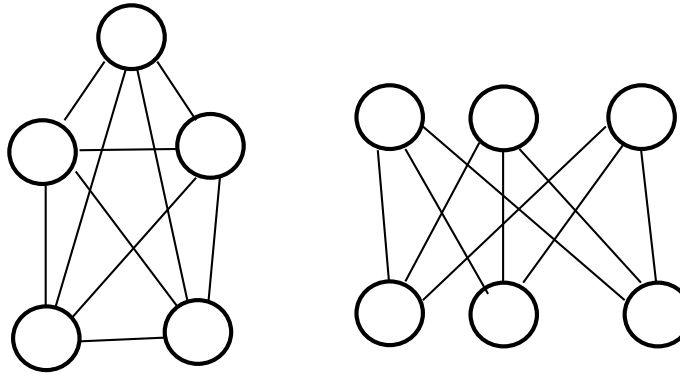
В справедливости формулы можно убедиться следующим образом. Граням сопоставим циклы из ребер, их ограничивающих. Число граней больше на единицу числа независимых циклов (за счет внешней грани), т. е. с учетом формулы для $n\,f - 1 = m - n + 1$, откуда непосредственно следует формула Эйлера.

Можно убедиться, что минимальным неплоским графом по числу элементов будет граф K_5 (рис. 11, *а*), по числу ребер — максимальный двудольный граф с числом вершин 6 (такой граф обозначается как $K_{3,3}$, рис. 11, *б*). Если такой граф входит в G, то G становится неплоским.

Граф G' называют гомеоморфным графу G, если он может быть получен из G заменой некоторых цепей на ребра.

Условие планарности графа определяет теорема Понтрягина–Куратовского:

граф G будет плоским тогда и только тогда, когда он не содержит графов, гомеоморфных графам K_5 и $K_{3,3}$.



а *б*

Рис. 11

Теорема Понтрягина–Куратовского определяет условия того, что граф является плоским, но не дает конструктивного способа организации проверки этого свойства (кроме полного перебора).

К практическим задачам, связанным с расположением графа на плоскости, можно отнести такие задачи, как

- расположение графа на плоскости с минимальным числом пересечений его ребер;
- выделение минимального числа ребер, удаление которых делает граф плоским;
- разбиение графа на минимальное число плоских подграфов.

4. ЭТАП ФУНКЦИОНАЛЬНО-ЛОГИЧЕСКОГО ПРОЕКТИРОВАНИЯ

4.1. Место и задачи этапа

На этом этапе готовится информация для рассмотренного выше конструкторского этапа исходя из описания алгоритмов выполнения различных команд вычислительно-управляющего устройства.

Алгоритм представляется в виде композиции двух моделей: граф-схемы алгоритма (ГСА), определяющего порядок выполнения операций, выделенных как элементарные, в операционном устройстве, и самого операционного устройства. В качестве примера таких операций можно назвать операцию сдвига содержимого мантиссы сумматора на заданное число, прием числа из памяти на регистр, передачу содержимого регистра на сумматор в прямом или обратном коде и др. По этой информации формулируются переключательные функции для реализации их в виде схем на конструкторском этапе.

Задание для рассмотренного выше конструкторского этапа формируется по ГСА выполнения отдельных операций и включает в себя решение двух связных между собой задач: получение автоматных описаний реализации алгоритмов в виде автоматных моделей и построение по ним описаний логических схем на заданных элементах памяти и в заданном базисе. Каждая схема представлена в виде системы переключательных (булевых) функций, которые должны быть получены на ее выходных полюсах. По этим функциям необходимо построить реализующую их схему, которая и будет заданием для конструкторского этапа проектирования.

4.2. Автоматные модели

4.2.1. Абстрактные автоматы

Определение. Абстрактным автоматом Мили называется преобразователь, представимый пятеркой [1]

$$S = <Z, W, A, \lambda, \delta>,$$

где Z — входной алфавит $\{z_1, z_2, ..., z_r\}$,

W — выходной алфавит $\{w_1, w_2, ..., w_p\}$,

A — Алфавит состояний автомата $\{a_1, a_2, ..., a_t\}$,

δ — функция переходов автомата: $AxZ \to A$,

λ — функция выходов автомата: $AxZ \to W$.

Если кроме того определено начальное состояние автомата a_0 (a_0 — символ из A), то говорят об инициальном автомате Мили.

Семантика: автомат функционирует в дискретном времени [1]. В любой момент времени он находится в некотором состоянии из множества A, и на его вход поступает входной сигнал из множества Z. В зависимости от состояния a_i и входного сигнала z_t, в следующий момент автомат переходит в новое состояние a_j, определяемое функцией перехода, $a_j = \delta(a_i, z_i)$, при этом на его выходе формируется выходной сигнал $w = \lambda(a_i, z_i)$ (рис. 12). В инициальном автомате в начале функционирования автомат находится в состоянии a_0.

Автомат Мили представляется двумя матрицами, переходов и выходов, описывающими соответствующие функции.

Пример 1. Автомат задан на множествах входов $Z = \{z_1, z_2, z_3\}$, состояний $A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$ и выходов $W = \{w_1, w_2\}$. Его функционирование описывается матрицами переходов и выходов, приведенными в табл. 1 и 2.

Таблица 1

Матрица переходов

	a_1	a_2	a_3	a_4	a_5	a_6
z_1	a_3	a_5	a_6	a_6	a_1	a_5
z_2	a_1	a_6	a_4	a_4	a_4	a_1
z_3	a_5	a_1	a_3	a_2	a_5	a_3

Таблица 2

Матрица выходов

	a_1	a_2	a_3	a_4	a_5	a_6
z_1	$w1$	w_1	w_2	w_1	w_2	w_1
z_2	$w1$	w_1	w_1	w_1	w_1	w_1
z_3	$w1$	w_1	w_2	w_2	w_2	w_1

Таблица 3

Объединенная матрица

	a_1	a_2	a_3	a_4	a_5	a_6
z_1	a_3/w_1	a_5/w_1	a_6/w_2	a_6/w_1	a_1/w_2	a_5/w_1
z_2	a_1/w_1	a_6/w_1	a_4/w_1	a_4/w_1	a_4/w_1	a_1/w_1
z_3	a_5/w_1	a_1/w_1	a_3/w_2	a_2/w_2	a_5/w_2	a_3/w_1

Автоматом Мура называют автомат, в котором функция выходов зависит только от состояния, то есть для него функция выходов автомата: μ: $A \to W$.

Пример 2. Автомат Мура задан на тех же множествах, что и автомат в примере 1, а его функционирование описывает таблице 4.

Таблица 4

Автомат Мура

	a_1/w_1	a_2/w_1	a_3/w_2	a_4/w_1	a_5/w_2	a_6/w_2
z_1	a_3	a_5	a_6	a_6	a_1	a_5
z_2	a_1	a_6	a_4	a_4	a_4	a_1
z_3	a_5	a_1	a_3	a_2	a_5	a_3

4.3. Структурные автоматы

4.3.1. Автоматная полнота и теорема В. М. Глушкова

В структурном автомате входные, выходные переменные, также как и состояния автомата, представляются не в абстрактном виде, как символы алфавита, а в виде наборов значений сигналов на входных и выходных полюсах автомата. Как правило, на этих полюсах действуют двоичные значения, обозначаемые как 0 и 1.

4. 3.2. Кодирование

Структурный автомат представляется в виде композиции комбинационной (логической) схемы и элементов памяти, связанных со схеме как показано на рис. 13. Входными переменными схемы являются входные переменные автомата — сигналы x_1, x_2, ..., x_n и переменные t_1, t_2,..., t_k, определяющие текущее состояние автомата, выходные переменные реализуются на выходах y_1, y_2,..., y_m. Выходы схемы $v1$, v_2, ..., v_k определяют переход автомата в следующее состояние.

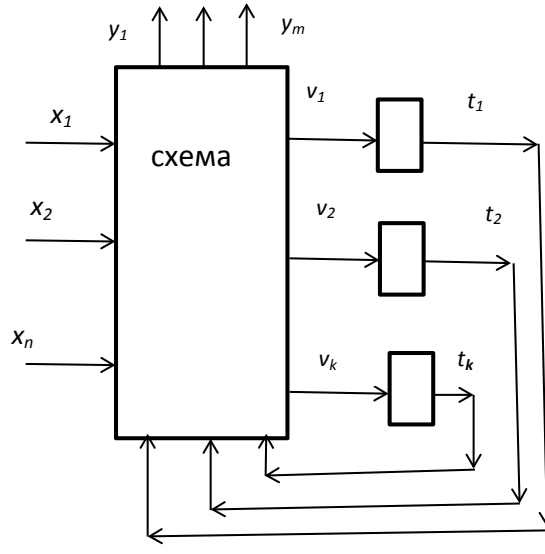


Рис. 13

Все переменные, приведенные на рис. 13, двоичные, то есть принимающие значение из множества {0,1}.

Переход от абстрактного автомата к структурному осуществляется через процедуру кодирования входов, выходов и состояний абстрактного автомата.

Кодирование входных переменных состоит в сопоставлении каждому символу входного алфавита абстрактного автомата набора

значений двоичных переменных $< x_1, x_2, ..., x_n >$ таким образом, чтобы каждый символ алфавита имел уникальный, отличный от других символов вектор. Чтобы это можно было сделать, число n должно быть таким, чтобы выполнялось условие $N \le 2^n$, где N – число символов входного алфавита.

Точно так же кодировка M символов выходного алфавита требует, чтобы число m обеспечивало равенство $M \le 2^m$, а кодировка K символов алфавита состояний была связана с числом k равенством $K \le 2^k$.

Во многих практических случаях кодировка входов и выходов абстрактного автомата уже задана семантикой входных и выходных сигналов, и тогда речь идет только о кодировании состояний. В общем случае можно считать, что кодировка может быть достаточно произвольной, тогда необходимо кодировать и входы с выходами.

Функции v_j называются функциями возбуждения элементов памяти, они должны переводить элементы памяти в состояния, определяющие следующее состояние. Вид этих функций зависит от того, какие элементы памяти используются. В счетном триггере при поступлении входного сигнала, равного нулю, элемент остается в том же состоянии, что и был. При единичном сигнале состояние элемента меняется с 0 на 1 и с 1 на 0.

В триггере типа «линия задержки» состояние автомата определяется только входным сигналом и не зависит от текущего состояния элемента: единичный входной сигнал устанавливает триггер в единичное состояние, нулевой сигнал — в нулевое состояние.

При кодировке могут использоваться различные критерии: минимальная схема автомата, обеспечение правильного функционирования автомата во времени с учетом временных характеристик элементов схемы и др.

4.3.3. Проектирование автомата

Проектирование автомата заключается в определении функций $\{y_i, v_j | i = 1, ..., m, j = 1, ..., k\}$ и синтезе в заданном базисе схемы, реализующей эти функции. Выходные функции определяются по матрице выходов автомата: после кодировки входов, выходов и состояний автомата матрица выходов представляет таблицу, описывающую m булевых выходных функций.

Функции возбуждения элементов памяти для случая триггера типа линии задержки также просто получаются из матрицы переходов автомата заменой входов и состояний автомата их кодами. Для счетного триггера эти функции получаются как результат операции сложения по модулю два кодировок текущего состояния и следующего состояния.

Таблица 5

Таблица переходов

	a_1	a_2	a_3	a_4
z_1	a_2	a_2	a_3	a_3
z_2	a_4	a_2	a_1	a_4
z_3	a_4	a_3	a_2	a_1

Пример: пусть автомат описывается матрицами переходов и выходов (табл. 5 и 6).

Таблица 6

Таблица выходов

	a_1	a_2	a_3	a_4
z_1	w_1	w_2	w_2	w_1
z_2	w_2	w_2	w_2	$w1$
z_3	w_1	w_1	w_1	w_2

Введем следующие кодировки для состояний, входов и выходов автомата (через переменные a_i, z_j, u, w_r): $a_1 = 00$, a_2=01, a_3=11, a_4=10, z_1= 00, z_2= 01, z_3= 11, w_1= 0, w_2= 1.

Таблица 7

Выходы

	00	01	11	10
00	0	1	1	0
01	1	1	1	0
11	0	0	0	1

Перепишем в этой кодировке матрицу выходов, получим описание выходных (в данном примере всего одной) функций (табл. 7).

Если обозначить кодирующие переменные входа как x_1 и x_2, состояний — как t_1 и t_2, выхода как y, то функция выхода будет иметь вид:

Таблица 8

Функции

$x \setminus t$	00	01	11	10
00	01	00	00	01
01	10	00	11	00
11	10	10	10	10

$$y = x_1 \bar{t}_1 \vee \bar{x}_1 \bar{t}_1 t_2 \vee x_1 \bar{x}_2 t_1 t_2 .$$

Если в качестве элемента памяти задан триггер типа линии задержки, то заменив в табл. 2.2 состояния на их коды, получим описание пары функций v_1 и v_2 (первый и второй столбец в табл. 8).

По этой таблице

$$v_1 = \bar{x}_1 \bar{x}_2 t_2 \vee \bar{x}_1 t_1 t_2 \vee x_1 \bar{t}_1 \bar{t}_2 \vee \bar{x}_2 \bar{t}_1 t_2 ,$$

$$v_2 = \bar{t}_1 \bar{t}_2 \vee \bar{x}_1 x_2 t_2 \vee x_2 t_1 t_2 .$$

4.4. Синтез схем

4.4.1. Определения

Пусть задано множество Y элементов, каждый из которых характеризуется числом входных полюсов и реализуемой этим элементом функцией от значений на входных полюсах. Значения переменных на входных и выходных полюсах элементов принимаются из множества {0,1}. Будем считать, что каждый элемент из Y определяет тип элемента и что в наличии есть неограниченное множество элементов данного типа.

Схемой назовем композицию элементов из Y, полученную отождествлением (связыванием) выходов некоторых элементов с входами других элементов, при котором выполняются следующие свойства:

1. С каждым входом связано не более одного выхода;
2. В схеме не образуются контуры;
3. С каждым выходом может быть связано более одного входа.

Назовем эти свойства свойством корректности связей схемы.

Все несвязанные входы элементов схемы назовем входами схем, а несвязанные выходы элементов и некоторые выделенные связанные выходы элементов схемы — выходами схемы.

Пример схемы приведен на рис. 14.

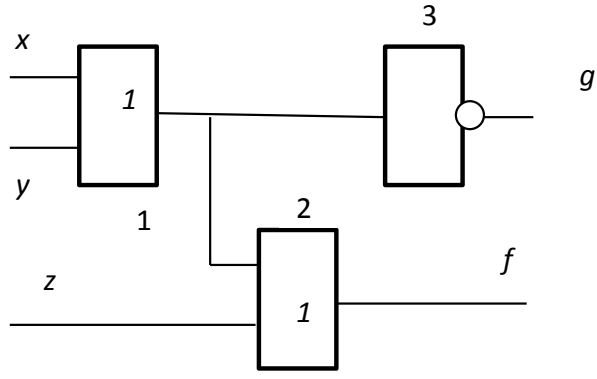


Рис. 14

Здесь x, y и z — входы схемы, g и f — ее выходы, элементы 1 и 2 реализуют функцию конъюнкции, элемент 3 – функцию инверсии.

Можно показать, что если выполняются условия корректности, то на выходах схемы реализуются функции, равные суперпозиции функций элементов схемы.

Для схемы 1: $f = (x \vee y) \vee z$, $g = x \vee y$.

Множество Y назовем базисом схемы.

Задачей синтеза схем назовем задачу построения для заданных функций $F = \{f_1, f_2, \ldots, f_m\}$ схемы в базисе Y, реализующих эти функции.

Будем оценивать решение числом элементов в схеме.

При реализации схем одной из первых задач является задача минимизации исходного описания схемы с целью получения решения, минимального по числу элементов.

Рассмотрим два подхода к синтезу схем — декомпозицию и факторизацию.

4.4.2. Метод декомпозиции

Метод декомпозиции еще называют методом от выходов к входам. Он состоит из последовательности шагов:

· определяется множество функций, называемых множеством непостроенных функций (обозначим как *F*н). Вначале *F*н = *F*;

· выбирается функция $F_н \in f$ и она удаляется из *F*н. Выбирается элемент $\gamma \subseteq \Psi$. Предполагается, что *f* реализуется на выходе элемента γ;

· определяются функции (называемые образующим списком), которые необходимо подать на вход элемент γ, чтобы на выходе его получить *f*;

· эти функции, за исключением входных переменных, вносятся в множество *F*н.

Шаги 2–4 повторяется до тех пор, пока множество *F*н не станет пустым. Для того чтобы получить решение, любая из функций образующего списка для *f* должна быть проще функции *f*. Самыми простыми считаются входные переменные, для других функций каждый метод предлагает свое понятие простоты.

Рассмотрим работу метода на примере реализации функции сложения по модулю два от двух переменных в одноэлементном базисе, состоящего из элемента 2-И-НЕ.

Таблица 9

a	*b*	*j*
0	0	1
0	1	1
1	0	1
1	1	0

Запишем таблицу для элемента базиса (табл. 9) и перепишем ее в виде табл. 10, где символом *x* обозначено значение «что угодно». Например, 2-я строка таблицы означает, что если на первом входе элемента нулевое значение, то, независимо от значения на втором входе, значение на выходе элемента равно единице.

Таблица 10

a	*b*	*j*
0	*x*	1
x	0	1
1	1	0

Запишем функцию в векторном виде как $f = a \oplus b =$<0110>.

При построении образующего списка будем исходить из того, что для реализации на выходе элемента 0 необходимо на оба входа подать 1, для реализации 1 необходимо на один вход подать 0, на другой — что угодно. Образующий список для *f* в этом случае будет иметь вид {<1x01>,<10x1>}.

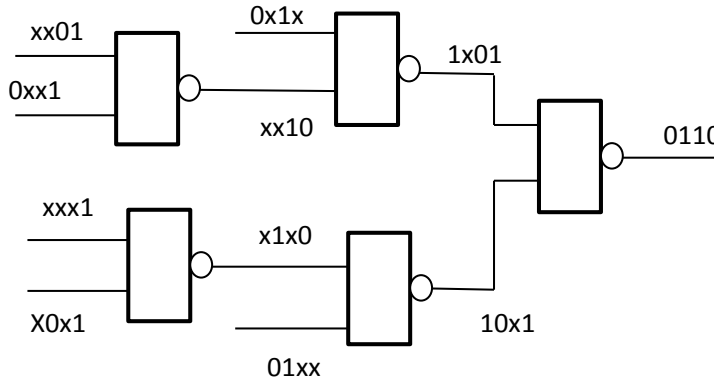


Рис. 15

Будем оценивать сложность функции числом неопределенных значений в ней. Будем при этом исходить из того, что чем больше неопределенность функции, тем легче доопределить ее до переменной. Из этой оценки следует, что вариант образующего списка {<1001>, <1xx1>} нас не устраивает, так как в этом варианте первая функция списка имеет ту же сложность, что и исходная.

Представим все решение в виде рис. 16. На выходе 2-го элемента верхняя функция дополняется до переменной b = <0011>, нижняя функция элемента 3 — до функции a = <0101>. На входе 5-го элемента нижняя функция дополняется до b, верхняя функция 4-го элемента — до a.

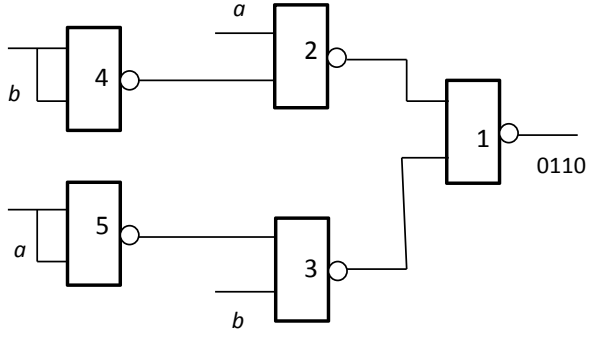


Рис. 16

К методу декомпозиции можно отнести и метод каскадов Поварова, основанный на разложении К. Шеннона. Функция представляется как $f(x_1,x_2,...x_n) = x \cdot f(x=1) \vee \bar{x} f(x=0)$, конструкция из элемента дизъюнкции и 2-х элементов конъюнкции образует каскад, на входе которого функции не зависят от переменной *x* и в этом смысле более просты, чем исходная функция.

В этом методе результат неоднозначен и зависит от порядка выбора переменных, по которым проводится разложение. Если функцию можно представить как сложение по модулю два переменной *x* и некоторой функции f^1 от остальных переменных, то $f^1 = f(x=1) = \overline{f}(x=0)$.

4.4.3. Метод факторизации

Метод факторизации заключается в алгебраических преобразованиях описаний заданных для реализации функций, с тем, чтобы выразить их через функции базиса. При этом важной частью преобразований является учет числа входов элементов базиса, что обеспечивается вынесением за скобки и использованием свойств дистрибутивности функций.

Рассмотрим применение метода для классического базиса, состоящего из элементов {2-И, 2-ИЛИ, НЕ}, и для монофункциональных базисов {И-НЕ} или {ИЛИ-НЕ.}

Синтез схем в классическом базисе

Решение задачи начинается с представления функции в виде ДНФ. Анализ функций направлен на выявление общих частей функций, которые должны будут реализовываться общими подсхемами. Это же относится и к реализации одной функции: общие части реализуются одной схемой, что значительно сокращает схему.

Рассмотрим применение метода на примере реализации пары функций $F = \{f1 = ab \vee a\bar{b}\bar{c} \vee \bar{a}\bar{b}\bar{c},\ f2 = abc\}$ в классическом базисе {2-И, 2-ИЛИ, НЕ}.

Прежде всего, минимизируем задание. После сокращения функции примут вид: $\{f1 = ab \vee \bar{a}\bar{b},\ f2 = abc\}$.

Учитывая вид второй функции, перепишем первую в виде

$f1 = ab \vee (\overline{a \vee b})$.

Теперь реализация первой функции потребует четыре элемента, второй функции — один элемент, с учетом того, что конъюнкция *ab* уже реализована в схеме первой функции (рис. 17).

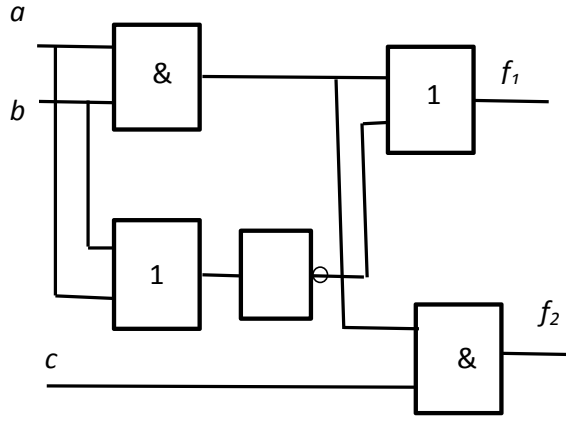


Рис. 17

При синтезе полезно использовать правило де Моргана. Действительно, если для реализации $f1 = (\bar{a}\,\bar{b})$ необходимы три элемента, то равная ей функция $f1 = \overline{(a \vee b)}$ потребует только два элемента.

Синтез схем в монофункциональном базисе

При синтезе схем в базисе Р-И-НЕ, где *P* — число входов элемента, используется следующий алгоритм.

Исходная функция представляется в дизъюнктивной нормальной форме, т. е. в виде $K1 \vee K2 \vee \ldots \vee Kr$, затем над ней проводится операция двойного отрицания, одно из которых опускается до конъюнкций, т. е. функция будет представлена через функции базиса.

Если $r \le P$ и число букв в каждой из конъюнкций не больше *P*, то задача решена. Схема будет состоять из двух уровней. На первом из них реализуются конъюнкции, на втором происходит их объединение. Задача усложняется, если на каком-то уровне входов у элемента не хватает для реализации. В этом случае задача состоит в том, чтобы разбить исходную запись функции на фрагменты и реализовать каждый из них отдельно, объединяя затем с использованием правила де Моргана.

Для этого проводится анализ конъюнкций в формуле, выделяются и выносятся за скобки общие части.

Этот же приём, связанный с вынесением за скобки и выделение общих частей, необходим, когда стоит задача реализации нескольких функций. При этом может оказаться, что для получения минимальной схемы не обязательно приводить все функции к минимальной форме.

Пример. Реализуем в базисе 3 -И-НЕ функцию f = <01101000>

Запишем функцию в СДНФ $f = \bar{a} \cdot \bar{b} \cdot c \vee \bar{a} \cdot b \cdot \bar{c} \vee a \cdot \bar{b} \cdot \bar{c}$

Функция не минимизируется, полученная нормальная форма является и минимальной.

Если возникает необходимость реализовать эту функцию в базисе 2-И-НЕ, то формулу преобразуем так:

$f = \bar{a} \cdot \bar{b} \cdot c \vee \bar{a} \cdot b \cdot \bar{c} \vee a \cdot \bar{b} \cdot \bar{c} = \bar{a}(\bar{b} \cdot c \vee b \cdot \bar{c}) \vee a(\bar{b} \cdot \bar{c}) =$

$= \bar{a} / (\bar{b} \cdot c \vee b \cdot \bar{c}) / (a / (\bar{b} \cdot \bar{c}))$.

Здесь символом / обозначена функция элемента базиса.

Инверсия реализуется на одном элементе базиса, если у него на оба входа подать одну и ту же переменную, т. е. $(x / x) = \bar{x}$.

При реализации на элементе Р-ИЛИ-НЕ используют представление функции в конъюнктивной нормальной форме. Все остальное — то же, что и при реализации в базисе Р-И-НЕ.

Пример. Реализуем ту же функцию в базисе 3-ИЛИ-НЕ.

$$f = (a \vee b \vee c) \cdot (a \vee \bar{b} \vee \bar{c}) \cdot (\bar{a} \vee b \vee \bar{c}) \cdot (\bar{a} \vee \bar{b} \vee c) \cdot (\bar{a} \vee \bar{b} \vee \bar{c}) =$$

$$= (a \vee b \vee c) \cdot (\bar{b} \vee \bar{c}) \cdot (\bar{a} \vee \bar{c}) \cdot (\bar{a} \vee \bar{b}).$$

5. МАТЕМАТИЧЕСКИЕ МОДЕЛИ ФУНКЦИОНАЛЬНО-ЛОГИЧЕСКОГО ЭТАПА

5.1. Булева алгебра

5.1.1. Основные определения

Булевой (логической) переменной называют переменную, принимающую значение из множества {0,1}. Название «логическая» следует из того, что ее значения трактуются чаще всего как «истина» (для 1) и «ложь» (для 0).

Функцией алгебры логики (переключательной или булевой функцией) от n переменных называют однозначное отображение множества всевозможных наборов значений n булевых переменных в множество {0,1}.

Такую функцию можно представить в виде таблицы из $n + 1$ столбцов и 2^n строк. Эта таблица называется таблицей истинности.

Таблица 11

x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Наборы значений переменных располагают в лексикографическом порядке (в порядке возрастания), как в примере в табл. 11 для $n = 3$.

Число всевозможных наборов значений переменных составляет $N = 2^n$. Число различных функций, которые могут быть записаны в таблице, равно 2^N.

Второй способ описания функции состоит в том, что перечисляются наборы значений, на которых функция равна 1 (множество $T1$) или 0 (множество $T0$).

Для приведенного примера функцию можно представить как $T1 = \{001,010,100, 111\}$.

Третий способ описания — представление функций в виде вектора. Так как порядок перечисления наборов входных переменных установлен, то достаточно указать только столбец функции. Для приведенного примера это будет вектор <01101001>.

Определение. Булева функция существенно зависит от переменной x_i, если найдутся два набора значений переменных, отличающиеся только i-й компонентой, на которых значения функции не совпадают. Переменная, от которой функция существенно не зависит, называется несущественной или мнимой для данной функции.

Простейшими называют функции от двух переменных. В табл. 12 приведены все функции, существенно зависящие от двух переменных. Для восьми из них введены названия и обозначения в табл. 13.

Таблица 12

Простейшие функции

$x\,y$	1	2	3	4	5	6	7	8	9	10
0 0	0	0	0	0	0	1	1	1	1	1
0 1	0	0	1	1	1	0	0	0	1	1
1 0	0	1	0	1	1	0	0	1	0	1
1 1	1	0	0	0	1	0	1	1	1	0

Таблица 13

Функция	Обозначение	Название
1	2	3
1	$x\&y$	Конъюнкция
4	$x \oplus y$	Сложение по модулю 2
5	$x \vee y$	Дизъюнкция

Окончание табл. 13

1	2	3
6	$x \uparrow y$	Стрелка Пирса (функция Вебба)
7	$x \approx y$	Эквивалентность
8	$y \to x$	Импликация из y в x
9	$x \to y$	Импликация из x в y
10	$x\|y$	Штрих Шеффера

Функция конъюнкции еще называется функцией И или логическим умножением и обозначается $x \& y = x \wedge y = x \cdot y = xy$.

Функцию дизъюнкции еще называют функцией ИЛИ (логическим сложением).

С помощью простейших можно строить более сложные функции, заменяя переменные функциями. В результате функции сопоставится формула, задающая последовательность выполнения операций. Для определения порядка вычисления функций можно использовать скобки.

Приведем свойства простейших функций, в истинности которых просто убедиться элементарной проверкой с помощью перебора.

Коммутативность дизъюнкции: $x \vee y = y \vee x$.

Это же свойство у конъюнкции и сложения по модулю 2.

Ассоциативность этих же функций: $(x \vee y) \vee z = x \vee (y \vee z) = x \vee y \vee z$.

Дистрибуция: $x \vee (yz) = (x \vee y)(x \vee z)$.

$x(y \vee z) = xy \vee xz$.

$x(y \oplus z) = (xy) \oplus (xz)$.

Правило де Моргана: $\overline{(x \vee y)} = \bar{x} \cdot \bar{y}$, $\overline{(x \cdot y)} = \bar{x} \vee \bar{y}$.

Свойства констант:

$$x \wedge 0 = 0, x \wedge 1 = x, x \vee 0 = x, x \vee 1 = 1, x \oplus 1 = \bar{x}, x \oplus 1 = 0, x1 = 0, x \vee \bar{x} = 1, x \wedge \bar{x} = 0.$$

Используя эти свойства, можно преобразовывать формулы, удаляя лишние элементы, раскрывая скобки, вынося элементы за знаки скобок и т. п.

5.1.2. Дизъюнктивные нормальные формы и теорема о разложении

Введем понятие степени булевой переменной. Будем считать, что $x^1 = x, x^0 = \bar{x}$.

Из определения следует, что $x^a = 1$, если $x = a$, и $x^a = 0$, если $x \neq a$. В справедливости этого можно убедиться простым перебором значений.

Конъюнкция $x_1^{a1} x_2^{a2} \ldots x_n^{an}$ называется элементарной, если в ней каждая переменная встречается не более одного раза.

Рангом элементарной конъюнкции называется число букв, образующих эту конъюнкцию.

Дизъюнктивной нормальной формой (ДНФ) называется дизъюнкция элементарных конъюнкций.

Длиной ДНФ называется число элементарных конъюнкций, образующих эту ДНФ. Длину ДНФ будем обозначать буквой L.

Дизъюнктивная нормальная форма, имеющая наименьшую длину по сравнению со всеми другими ДНФ, эквивалентными данной функции, называется кратчайшей ДНФ (КДНФ).

Дизъюнктивная нормальная форма, содержащая наименьшее число букв x_i^{ai} по сравнению со всеми другими ДНФ, эквивалентными данной функции, называется минимальной ДНФ (МДНФ).

Теорема о разложении. Для любой функции $f(x_1, x_2, \ldots, x_n)$

$$f(x_1, x_2, \ldots, x_i, \ldots, x_n) = \bigcup_{\forall a_1 a_2 \ldots a_i} x_1^{a1} x_2^{a2} \ldots x_i^{ai} f(a_1, a_2, \ldots, a_i, x_{i+1} \ldots, x_n).$$

Доказательство. Возьмем произвольный набор значений переменных $(b_1, b_2, \ldots, b_n)$ и подставим его в выражение справа и слева от равенства. Получим

$$f(\beta_1, \beta_2, \ldots, \beta_i, \ldots, \beta_n) = \bigcup_{\forall a_1 a_2 \ldots a_i} \beta_1^{a1} \beta_2^{a2} \ldots \beta_i^{ai} f(a_1, a_2, \ldots, a_i, \beta_{i+1} \ldots, \beta_n).$$

Справа для любого набора значений $(a_1, a_2, \ldots, a_i)$, не равного $(b_1, b_2, \ldots, b_i)$, соответствующая конъюнкция будет равна нулю, так как она, в силу условия $x^a = 0$, если $x \neq a$, будет содержать нулевой сомножитель. Останется единственная конъюнкция, где $(a_1, a_2, \ldots, a_i) = (b_1, b_2, \ldots, b_i)$, т. е. получим равенство $f(b_1, b_2, \ldots, b_n) = f(b_1, b_2, \ldots, b_n)$, что и доказывает теорему.

Следствия из теоремы:

1. $f(x_1, x_2 \ldots) = x_i f(x_i = 1) \vee \bar{x}_i f(x_i = 0)$. Это равенство известно как формула разложения К. Шеннона по переменной x_i.

2. $f(x_1, x_2, \ldots, x_n) = \cup x_1^{a1} \ldots x_n^{an} f(a_1, a_2, \ldots, a_n) = \bigcup_{\forall a1, a2, \ldots an \in T1} x_1^{a1} x_2^{a2} \ldots x_n^{an}$.

Это равенство получается из формулы при $i = n$, здесь Т 1 — множество наборов значений аргументов, на которых функция равна 1. Оно известно как представление функции в виде совершенной дизъюнктивной нормальной формы (СДНФ).

Совершенная дизъюнктивная нормальная форма состоит из элементарных конъюнкций ранга n, т. е. конъюнкций наибольшего возможного для данной функции ранга, поэтому с этой точки зрения СДНФ является наиболее сложной.

Любую функцию можно представить в виде СДНФ, и это представление единственно, так как оно однозначно сопоставляется таблице истинности функции.

5.1.3. Минимизация в классе ДНФ

Во всех методах минимизации функций, представленных в каноническом базисе, используются следующие операции.

1. $f \cdot a \vee f \cdot \bar{a} = f$.

Здесь f — некоторая формула. Это равенство поясняется следующим образом: $f \cdot a \vee f \cdot \bar{a} = f = f(a \vee \bar{a}) = f$. При этом используется свойство дистрибуции операций конъюнкции и дизъюнкции, свойство констант $a \vee \bar{a} = 1$, $f \cdot 1 = f$. Формула представляет операцию, называемую операцией склеивания по переменной, в данном случае переменной a. Использование этой операции при минимизации функций сокращает число элементов в ней на 4+|f|, где |f| — число переменных в функции f.

Пример: $\bar{a}\bar{b}\bar{c} \vee a\bar{b}\bar{c} = \bar{b}\bar{c}$. В исходной функции используется 10 элементов (5 инверсий, 4 конъюнкции и 1 дизъюнкция), в результате — 3 элемента (2 инверсии, 1 конъюнкция). В функции $f = \bar{b}\bar{c}$ 3 элемента, сокращение на 10−3=7 элементов.

2. $fa \vee f = f$.

Это равенство следует из того, что функция f уже содержит $f \cdot a$ ($f = f \cdot a \vee f \cdot \bar{a}$). Эта операция называется операцией поглощения. Сокращение в числе элементов при этом составляет 2+|f| элементов.

3. $f \cdot a \vee \bar{a} = f \vee \bar{a}$.

Справедливость этого равенства следует из следующего:

$fa \vee \bar{a} = fa \vee f\bar{a} \vee \bar{f}\bar{a} = fa \vee f\bar{a} \vee \bar{f}\bar{a} = f \vee \bar{a}$.

Сокращение при использовании этой операции —1 элемент.

Во многих случаях сложность ДНФ оценивают числом входящих в нее букв.

5.1.4. Тупиковые нормальные формы

Любая функция алгебры логики может быть записана в виде СДНФ [1]. Запись в СДНФ является самой сложной из ДНФ.

Пример. Пусть задана СДНФ:

$\mathrm{f}(a,b,c) = abc \vee ab\bar{c} \vee a\bar{b}c \vee a\bar{b}\bar{c} \vee \bar{a}bc$.

Преобразуем эту СДНФ. Добавим еще один конъюнктивный член *abc*. Это добавление не меняет данной функции, так как $x \vee x = x$,

$f(a,b,c) = abc \vee ab\bar{c} \vee a\bar{b}c \vee a\bar{b}\bar{c} \vee abc \vee \bar{a}bc$.

Преобразуем это выражение, используя описанные в предыдущем разделе операции. Получим:

$f(a,b,c) = ab(c \vee \bar{c}) \vee a\bar{b}(c \vee \bar{c}) \vee bc(a \vee \bar{a}) = ab \vee a\bar{b} \vee bc$.

Аналогично предыдущему делаем дальнейшие преобразования:

$f(a,b,c) = a(b \vee \bar{b}) \vee bc = a \vee bc$.

Если к исходной ДНФ применять в произвольном порядке описанные в предыдущем разделе преобразования, то наступит момент, когда эти преобразования окажутся уже невозможными. В этом случае получим ДНФ, которую называют тупиковой ДНФ (ТДНФ).

Минимальная ДНФ содержится среди ТДНФ. Получая всевозможные тупиковые ДНФ и сравнивая их по числу букв, можно найти минимальную формы для данной функции. Величина необходимого перебора определяется количеством функций в классе тупиковых ДНФ. Для функции алгебры логики, зависящей от числа аргументов, больше 10 переборов может быть очень большой. Поэтому на практике часто ограничиваются в этом случае получением первой тупиковой формы.

Метод минимизации записанной в ДНФ функции, основанный на алгебраических преобразованиях с использованьем операций из раздела 5.1.3, назовем алгебраическим.

При минимизации удобно использовать описание функции в виде перечисления множества значений Т 1 — множество единич-

ных наборов. Номера наборов необходимо записывать в виде двоичных номеров. Так, функцию, приведенную в начале данного раздела, представим в виде Т 1 = {111, 110, 101, 100, 011}. Две конъюнкции можно склеить только тогда, когда им составлены двоичные наборы, отличающиеся только в одной позиции — значением одной переменной. Такие наборы называются соседними. Соседние наборы склеиваются по той переменной, значения которой в наборах различны. В нашем примере соседними будут, например, наборы 111 и 110, что соответствует конъюнкциям abc и $ab\bar{c}$. Полученную в результате склеивания конъюнкцию обозначим как 11~, где символом «~» обозначено отсутствие переменной *c*, или, что то же самое, произвольное значение этой переменной (хоть 0, хоть 1).

5.1.5. Метод минимизации по картам Карно

Данный метод минимизации применим для функций с числом переменных не более 6 и удобен для ручной минимизации, когда человек видит те комбинации, которые можно объединить вместе. Для этого запишем таблицу в виде матрицы размером $n_1 \times n_2$, где все переменные разбиты на две по возможности, равные группы, n_1 и n_2 соответственно, множества всевозможных наборов значений переменных первой и второй группы. Эти наборы упорядочим таким образом, чтобы соседние наборы были расположены рядом.

Пусть функция зависит от четырех переменных — *a*, *b*, *c* и *d*. Сопоставим столбцам наборы значений переменных *a* и *b*, строкам — переменных *c* и *d,* так, чтобы каждой клетке соответствовала комбинация переменных из этих групп и соседние наборы расположились рядом. Соседние наборы для рассматриваемого случая: 00→01→11→10 (при каждом последующем переходе изменяется только подчеркнутый символ). При этом первый и последний наборы также оказываются соседними. Эта таблица называется картой Карно.

В качестве примера рассмотрим функцию, заданную множеством единичных наборов *T*1 = {00, 01, 03, 05, 07, 10, 11, 12, 16, 17}, где значение наборов представлено в восьмеричном виде. Переведем эти значения в двоичный вид: 0000, 0001, 0011, 0101, 0111, 1000, 1001, 1010, 1110, 1111.

Карта Карно приведена в виде табл. 14.

Таблица 14

cd \ *ab*	00	01	11	10
00	1_1			1_6
01	1_2	1_4		1_7
11	1_3	1_5	1_9	
10			1_{10}	1_8

Заполнение карты производится по табличному описанию исходной функции. В примере конъюнкции 0000 соответствует клетка 00/00, а 0111 — клетка 01/11 и т. д. В данной таблице каждая единица имеет порядковый индекс, который соответствует порядковому номеру данной компоненты в исходной функции (расстановка этих индексов совершенно не обязательна и здесь приведена для лучшего понимания).

Для минимизации необходимо попарно «склеить» рядом стоящие единицы, имеющие хотя бы одну общую компоненту. Общими будут компоненты, заполняющие прямоугольник размером $2^i \times 2^j$. При этом нужно не забывать, что крайние строки и крайние столбцы также соседние, поэтому можно объединять, например, ячейки с единицами $1_1, 1_2, 1_6$ и 1_7, результатом чего будет конъюнкция $\bar{b}\bar{c}$. Действительно, после объединения 1_1 и 1_2 получим конъюнкцию $\bar{a}\bar{b}\bar{c}$ (от *d* не зависит, так как эта переменная принимает значение как 1, так и 0), объединение $1_6 1_7$, дает конъюнкцию $a\bar{b}\bar{c}$, которая склеивается с первой конъюнкцией и дает результат` $\bar{b}\bar{c}$. Нужно стремиться «склеить» в один набор как можно больше клеток. Склеиваем далее $1_2, 1_3, 1_4$ и 1_5, получаем конъюнкцию $\bar{a}d$. 1_8 можно склеить с 1_6 или с 1_{10}. В первом случае получим конъюнкцию $a\bar{b}\bar{d}$, во втором случае — $ac\bar{d}$. Предпочтение отдаем второму решению, где меньше инверсий. Остается непокрытой только 1_9. Здесь также два варианта — склеить ее с 1_{10} или с 1_5. В первом случае конъюнкция не будет зависеть от переменной *d* и будет равна *abc*, во втором случае не будет зависеть от переменной *a* и будет равна *bcd*. Оба случая равнозначны. В результате получим формульное представление функции:

$$f = \bar{b}\bar{c} \vee \bar{b}d \vee ac\bar{d} \text{ или}$$

$$f =` \bar{b}\bar{c} \vee \bar{b}d \vee ac\bar{d} \vee bcd .$$

С точки зрения сложности ДНФ оба представления имеют одну и ту же сложность.

Как указывалось выше, сложность формульного представления можно оценивать числом символов переменных в формуле. Для полученного решения эта оценка будет равной 10. Будем в дальнейшем пользоваться такой оценкой.

Для функций от пяти и шести переменных, когда столбцы для пяти переменных и столбцы и строки для шести переменных именуются наборами значений трех переменных. В этом случае порядок может быть таким: 000→001→011→010→110→111→101→100. Снова крайние наборы соседние, любую пару соседних наборов тоже можно склеить. Однако склеить четыре подряд расположенных набора возможно не всегда. При склеивании четырех наборов должны сократиться две переменные, т. е. во всех этих наборах только одна переменная должна сохранять постоянное значение. Например, для наборов 000, 001, 011 и 010 такой переменной является первая переменная. Если эти переменные *a*, *b* и *c*, то этим наборам будет сопоставлена (вырожденная) конъюнкция $\bar{a}$. Для рядом расположенных наборов 001, 011, 010 и 110 такой переменной не находится, и значит, их склеить нельзя. Склеивая их попарно, получим две конъюнкции: $\bar{a}c$ и $b\bar{c}$, которые, как видно, соседними не являются. Это первая сложность при минимизации функций пяти и шести переменных.

Вторая сложность состоит в том, что некоторые не соседние наборы являются соседними, и их можно склеивать. Примером может служить множество наборов 000 и 010, наборы 001 и 101, наборы 011 и 111. Четыре набора 001, 011, 111 и 101 также склеиваются в набор, соответствующий конъюнкции *c*.

Пример. Минимизируем функцию пяти переменных:

$$f(a,b,c,d,e) = \underset{1}{a\cdot b\cdot\bar{c}\cdot\bar{d}} \vee \underset{2}{a\cdot\bar{b}\cdot d\cdot\bar{e}} \vee \underset{3}{\bar{a}\cdot\bar{b}\cdot c\cdot d} \vee \underset{4}{a\cdot\bar{b}\cdot c\cdot\bar{e}} \vee \underset{5}{a\cdot\bar{b}\cdot\bar{c}\cdot e} \vee \underset{6}{a\cdot b\cdot\bar{c}\cdot d}\ .$$

Карта Карно для нее приведена в табл. 15.

Таблица 15

*de**abc*	000	001	011	010	110	111	101	100
00							1_4	1_1
01								1_{15}
11		1_3			1_6			1_5
10		1_3			1_6		1_{24}	1_2

Если в конъюнкции переменная не присутствует, то 1 ставится во все клетки, удовлетворяющие присутствующим переменным. Так, например, первой конъюнкции ($a \cdot b \cdot \bar{c} \cdot \bar{d}$) соответствуют две клетки: 100/00 и 100/01 с различными значениями переменной *e*, что обозначено как $1_{1,5}$ и 1_1, четвертой конъюнкции $a \cdot \bar{b} \cdot c \cdot \bar{e}$ – клетки 101/00, 101/10.

Минимизация приводит к формуле

$$f(a,b,c,d,e) = \underset{1}{a \cdot \bar{b} \cdot \bar{c}} \vee \underset{2}{a \cdot \bar{b} \cdot \bar{e}} \vee \underset{3}{\bar{a} \cdot \bar{b} \cdot c} \vee \underset{4}{a \cdot \bar{c} \cdot d} .$$

Соответствующие покрытия показаны в табл. 16.

Таблица 16

de\abc	000	001	011	010	110	111	101	100
00							12	112
01								11
11		13			14			114
10		13			14		12	1124

5.1.6. Минимизация частичных функций

Функция называется частичной, или неполностью определенной, если ее значения могут быть не только 0 и 1, но и неопределенными, обозначаемыми в таблице и в картах Карно символом «–» или «~». Будем пользоваться первым символом.

Семантика понятия «неопределенность» двояка. В первом случае неопределенность предполагает, что на практике такой набор значений входных переменных не может встретиться. Во второй трактовке неопределенность означает, что при заданном наборе значений входных переменных значение функции может быть любым, так как он нигде не используется. В обоих случаях частичной функции сопоставляется 2^k различных полностью определенных функций, где k — число неопределенных наборов значений входных переменных. Значит, необходимо из этих функций выбрать такую, которая имеет минимальную сложность. Частичной функции сопоставляется, кроме множеств значений $T1$ или $T0$, еще множество неопределенных значений $T\sim$, и для определения функции необходимо задать любые два множества из этих множеств, например $T1$ и $T\sim$.

Таблица 17

a	*b*	*c*	*d*
0	0	0	~
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	~
1	1	0	0
1	0	1	1

Пример. Частичная функция определена табл. 17. Эта функция может быть реализована одной из $2^2 = 4$ полностью определенными функциями. Единичным значениям функций в СДНФ будут сопоставлены конъюнкции $f(a,b,c,d,e) = \bar{a}\cdot\bar{b}\cdot c, \bar{a}\cdot b\cdot\bar{c}, a\cdot b\cdot c$ или в «двоичном представлении» — 001, 010, 111. Неопределенности представлены конъюнкциями $\bar{a}\cdot\bar{b}\cdot\bar{c}$ и $a\cdot b\cdot c$ или 000 и 111. Видно, что единичные наборы не являются соседними между собой, но первую конъюнкцию можно сократить на переменную *c*, если склеить ее с первой неопределенной конъюнкцией. С последней неопределенной конъюнкцией можно склеить и последнюю определенную. При этом из конъюнкции удалится переменная *b*. Вторая определенная конъюнкция также склеивается с первой неопределенной, в результате чего удаляется переменная *b*. Результатом будет функция $\bar{a}\cdot c \vee \bar{a}\cdot\bar{c} \vee a\cdot c$, или в векторном виде <11100101>. Вторым вариантом будет вариант, когда первая конъюнкция описания функции склеивается с последней конъюнкцией неопределенных значения, тогда результатом будет формула $\bar{a}\cdot\bar{b} \vee \bar{a}\cdot\bar{c} \vee a\cdot c$.

5.1.7. Метод Квайна—Мак-Класки

Рассматриваемый метод пригоден для любого числа переменных (практически не более 10), допускает возможность построения компьютерной программы минимизации.

Рассмотрим сначала случай, когда функция представляется в виде множества единичных наборов *T*1, каждая конъюнкция которой переведена в двоичный вектор. Заметим, что склеивание можно проводить только между наборами, которые отличаются между собой только в одном разряде, и значит, число единиц в этих наборах будет отличаться на единицу. Например, набору $\bar{a}\cdot b\cdot\bar{c}\cdot d\cdot e$ соответствует вектор 01011, в котором три единицы, а соседнему с ним по переменной *a* набору $a\cdot b\cdot\bar{c}\cdot d\cdot e$ – вектор 11011 с числом единиц 4.

На первом этапе разделим все наборы на группы по числу единиц в них. Согласно сказанному выше, склеиваться могут только наборы из соседних групп, поэтому последовательно проводим все склеивания, которые только возможны. В результате этой операции на втором уровне получаем вектора в троичном представлении, где используется символ «~» как символ неопределенного значения. Так после склеивания конъюнкций, приведенных в начале параграфа, на втором уровне получим вектор ~1011. В результате проведенных склеиваний на втором уровне после склеивания наборов из групп, содержащих i и $(i+1)$ единиц, получим группу, все элементы которой содержат i единиц. На следующих этапах снова сравниваем вектора из соседних групп, склеиваем отличающиеся только в одной позиции вектора. Это отличие может быть лишь в том, что в некотором разряде одного вектора стоит 0, в этом же разряде второго вектора стоит 1, а все остальные разряды совпадают полностью, с учетом и символов «~». Так, если на втором уровне в группе с тремя единицами получили вектор ~1011, а в группе с числом единиц 2 получили вектор ~1010, то их склеиваем и получаем на третьем уровне в группе с двумя единицами вектор ~101~. Эту операцию продолжаем, пока возможно. В результате на r-м уровне получим вектора, число символов «~» в которых равно $(r-1)$. Такую операцию проводим, пока возможно.

Пример

Функция от 4-х переменных представлена как Т 1 = {0011, 0100, 0101, 0111, 1001, 1101, 1110, 1111} .

Разобьем конъюнкции, представленные наборами, на группы в зависимости от числа единиц в представлении (рис. 18).

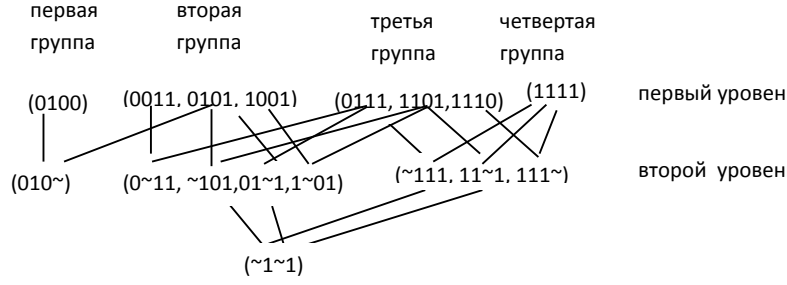


Рис. 18

Здесь подчеркнуты конъюнкции, которые не «ушли» на следующий уровень.

На следующем этапе строим таблицу покрытия. Столбцы таблицы именованы векторами, сопоставленными исходному описанию функции. Для рассматриваемого примера таблица имеет следующий вид (табл. 18).

Таблица 18

	0011	0100	0101	0111	1001	1101	1110	1111
010~		*v*	*v*					
0~11	*v*			*v*				
1~01					*v*	*v*		
111~							*v*	*v*
~1~1			*v*	*v*		*v*		*v*

Необходимо выбрать минимальное число строк, покрывающих все столбцы. Начинается решение задачи с выбора столбцов, содержащих единственную отметку. Таким является столбец 0011: в решении необходимо взять конъюнкцию 0~11, иначе конъюнкция 0011 покрыта не будет (не войдет в решение). Но выбор конъюнкции 0~11 покрывает и конъюнкцию 0111. Отмечаем эти два столбца. Таким же образом поступаем с конъюнкциями 0100, 1001 и 1110, которые требуют в решение, соответственно, конъюнкции 010~, 1~01 и 111~. Но первая покрывает еще и 0101, вторая — 1101, третья — 1111. В результате все исходные конъюнкции оказываются покрытыми. Решение найдено, функция будет иметь минимальную ДНФ в виде

$f = \bar{a} \cdot c \cdot d \vee \bar{a} \cdot b \cdot \bar{c} \vee a \cdot \bar{c} \cdot d \vee a \cdot c \cdot d$.

Заметим, что в этом примере самая короткая конъюнкция (~1~1), покрывающая наибольшее число исходных конъюнкций, в решение не вошла.

5.1.8. Минимизация системы функций

Минимизация в рамках дизъюнктивных нормальных форм является первым этапом в нахождении простейшего представления функций для их последующей реализации в виде схем, когда элементам схемы сопоставляются функции конъюнкции (элемент ИЛИ), дизъюнкции (элемент И) и инверсии (элемент ИНВЕРТОР). Даль-

нейшая минимизация связана с получением скобочных форм, когда находятся в схеме общие части, которые выносятся за скобки с учетом свойств дистрибуции конъюнкции относительно дизъюнкции и дизъюнкции относительно конъюнкции.

Рассмотрим в качестве примера последнюю функцию, полученную после минимизации методом Квайна–Мак-Класки:

$f = \bar{a} \cdot c \cdot d \vee \bar{a} \cdot b \cdot \bar{c} \vee a \cdot \bar{c} \cdot d \vee a \cdot c \cdot d$.

Сложность этой функции (в числе символов переменных) равна 12. Однако если из первых двух конъюнкций вынести общий сомножитель $\bar{f}_2$, а из двух последних — a, то получим формулу $f = \bar{a} \cdot (c \cdot d \vee b \cdot \bar{c}) \vee a \cdot (\bar{c} \cdot d \vee c \cdot d)$.

Сложность этой формулы будет уже 10.

В общем случае может существовать множество возможностей таких преобразований, из которых необходимо выбрать вариант, имеющий минимальную сложность.

5.1.9. Системы функций

Очень часто схема должна реализовать не одну функцию, а систему функций. В этом случае желательно получить минимальное представление обо всей системе, в которой используется минимальное число используемых операторов. Здесь существуют различные способы минимизации. Во-первых, при представлении одной функции нужно учитывать возможность использовать подформулы, полученные в формулах, при представлении других функций.

Пример. Система функций содержит две функции:

$F = \{ f_1 = \bar{a} \cdot b \vee \bar{b} \cdot a \ , \ f_2 = a \cdot b \cdot c \}$.

Преобразуем первую функцию, конъюнкцию 3 используем во второй функции:

$$f_1 = (a \vee b) \cdot (\bar{a} \vee \bar{b}) = (a \vee b) \cdot \overline{(a \cdot b)}, f_2 = (a \cdot b) \cdot c$$

Сложность оценивается в пять элементов: три конъюнкции, одна дизъюнкция и одна инверсия (конъюнкция *ab* используется в обеих функциях).

Второй способ связан с тем, что проводится анализ функций на предмет того, можно ли получить из одной функции другую с помощью некоторых простых операций. В качестве таких операций

можно использовать, например, умножение на переменную или ее инверсию.

Пример:

$$f_1 = (a \cdot \bar{b} \cdot c \vee a \cdot (b \to \bar{d})),$$
$$f_2 = (\bar{a} \cdot \bar{b} \cdot \bar{c} \vee ((a \vee b) \to b \cdot \bar{c}).$$

После минимизации функций получим:

$$f_1 = (a \cdot \bar{b} \cdot c \vee a \cdot (\bar{b} \vee d)),$$
$$f_2 = (\bar{a} \cdot \bar{b} \cdot \bar{c} \vee ((\bar{a}\bar{b}) \vee b \vee \bar{d}) = (\overline{(a \vee b)} \vee b\bar{d}).$$

Реализация этой пары функций потребует: для первой функции три элемента (одну конъюнкцию, одну инверсию и одну дизъюнкцию), для второй функции — пять элементов (одну конъюнкцию, две инверсии и две дизъюнкции).

Но есть решение, требующее только шесть элементов.

Реализация f_1 требует четыре элемента: две дизъюнкции, одну конъюнкцию и одну инверсию. Заметим, что инверсия $\bar{f}_1 = a \cdot \bar{b} \vee a \cdot d \vee b \cdot d = a \cdot (\bar{b} \vee d) \vee b \cdot d.$

Получим конъюнкцию ее с переменной a:

$$a \cdot \bar{f}_2 = a \cdot (a \cdot \bar{b} \vee a \cdot d \vee b \cdot d) = a\bar{b} \vee a \cdot d \vee a \cdot b \cdot d = a \cdot (\bar{b} \vee d) = f_1.$$

Таким образом, если построить $\bar{f}_2$, (четыре элемента), затем с помощью умножения на x (еще один элемент) получить f_1, а через инверсию (еще один элемент) получить f_2, решение будет содержать шесть элементов!

6. ЭТАП АРХИТЕКТУРНОГО ПРОЕКТИРОВАНИЯ

6.1. Место и задачи этапа

Этап архитектурного проектирования является первым этапом в процессе проектирования. В различных предметных областях этот этап еще носит название блочного или структурного проектирования. На этом этапе исходным заданием для решения является техническое задание на проектируемое устройство. Прежде всего, исходя из класса решаемых задач происходит выбор набора команд, наиболее подходящих для проектируемого вычислительного или управляющего устройства. Формальных методов выбора нет или они слишком громоздки, поэтому используется такой подход: предлагается решение, которое проверяется на соответствие заданным условиям (производительность, необходимое время, корректность и др.). После этого проводится анализ алгоритмов для команд, которые будут выполняться на проектируемом устройстве, выбирается оптимальное множество команд с учетом времени выполнения команды, частоты ее исполнения и других параметров. Например, нужно ли вводить команду извлечения квадратного корня или ее можно заменить подпрограммой через другие команды? Здесь выбор не формализован и зависит от многих условий. В итоге на этап проектирования приходит множество алгоритмов, которые должны быть реализованы в проектируемом устройстве.

При реализации алгоритма используется модель устройства как композиции двух блоков: управляющего устройства (УУ) и операционного (арифметического) устройства (АУ). Алгоритм УУ записан через операции, выполняемые в АУ, например операции очистки регистров, запись на них данных из памяти и передачу содержимого регистра на сумматор и др. В УУ из АУ приходят сигналы о значении ячеек АУ (знак числа, знак порядка, значение старшего разряда сумматора (для решения вопроса о нормализации)). Очевидно, в АУ может выполняться одновременно несколько операций (например, прием

числа на первый регистр, очистка сумматора). Одной из центральных задач является задача проверки корректности алгоритмов [3].

Будем пользоваться представлением алгоритма в виде граф схемы алгоритма (ГСА). ГСА определяется как композиция вершин следующих типов: начальная вершина (*Begin*), конечная вершина (*End*), операторная вершина (*Process*,) условная вершина (*If*), сборка по ИЛИ (*U*). Семантика такого представления и способ изображения этих вершин на ГСА общеизвестны — с алгоритмом связано понятие управления, которое начинается в вершине *Begin*, происходит в вершинах *Process*, передается от вершины к вершине и заканчивается, когда управление приходит в вершину *End*. Таким образом, преобразованию соответствует последовательность (слово) в алфавите *P*, которая определяется последовательностью выполнения условных операторов.

Параллельные ГСА используют при описании еще два типа вершин — сборку по И (*Join*), когда продолжение алгоритма возможно, если по всем заходящим в эту вершину дугам пришло управление, и вершина типа разветвления (*Fork*), после которой управление передается одновременно по всем исходящим из нее дугам.

Определение. ПГСА корректна, если после начала выполнения алгоритм приходит при всех возможных условиях в конечную вершину, и при этом все процессы в алгоритме завершаются. Одной из главных задач является проверка ПГСА на корректность [4].

6.2. Корректность ПГСА. Методы проверки корректности

6.2.1. Анализ слов, порождаемых ПГСА

ПГСА в процессе работы порождает слова в алфавите: $A` = \{B\} \cup \{Ps\} \cup \{Is\} \cup \{Us\} \cup \{Js\} \cup \{Fs\} \cup \{E\}$,

где $\{B\}$ и $\{E\}$ — начальная и конечная вершины алгоритма,

$\{P_s\}$ — множество операторных вершин,

$\{I_s\}$ — множество условных вершин (*If*),

$\{U_s\}$ — множество вершин сборки по ИЛИ,

$\{J_s\}$ — множество вершин сборки по И (*Join*),

$\{F_s\}$ — множество вершин типа разветвления (*Fork*).

Слово $A = A_1 \dots A_i A_{i+1} \dots A_к$ допустимо, если оно начинается с вершины типа B ($A_1 = B$) и заканчивается вершиной типа E ($A_к = E$), и при этом для любого i A_i не равно A_{i+1}.

Пусть $V = \{B\} \cup \{E\} \cup \{P\} \cup \{U\} \cup \{I\} \cup \{F\} \cup \{J\}$ — множество вершин ГСА, а *Vi* — вершина из этого множества.

Еще одно требование к слову *A* определяет семантика исполнения команд. Должны выполняться в зависимости от команд следующие отношения:

1) отношение строгого порядка $Vi < Vj$, если от вершины *Vi* к вершине *Vj* есть путь S_{ij}, а пути S_{ij} не существует;

2) отношение циклического порядка $Vi \Leftrightarrow Vj$, если есть путь и от вершины V*i* к вершине V*j* — S_{ij}, и S_i – от *Vj* к *Vi*;

3) отношение неопределенного порядка $Vi \,//\, Vj$, путей S_{ij} и S_{ij} не существует, однако есть вершина *Vк* из {*F*} (множество вершин типа «ветвление»), такая, что существуют пути $S_{кj}$ и $S_{кj}$;

4) отношение взаимного исключения $Vi \,/\, Vj$, нет путей S_{ij} и S_{ij}, но есть вершина $V_к$ (выбор в зависимости от условия), такая, что существуют пути $S_{кj}$ и $S_{кj}$.

Метод проверки корректности связан с перебором всех возможных ситуаций при работе алгоритма.

6.2.2. Анализ структуры ГСА

Рассмотрим методы проверки корректности с помощью анализа ПГСА. Существует три вида «плохих» фрагментов, наличие которых свидетельствует о некорректности ПГСА.

1) Дедлок — ситуация, когда несколько вершин (чаще две) ожидают завершения вычислений друг друга, например как на рис. 19.

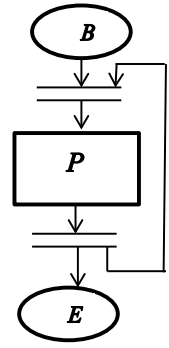


Рис. 19

2) Недетерминированность — приход в вершину *E* по одному из возможных путей, когда в остальных путях вычисления продолжаются. Пример на рис. 20.

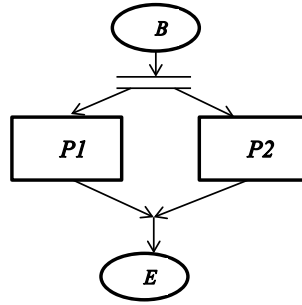


Рис. 20

3) Зависание — ожидание того, что никогда не сможет произойти (рис. 21).

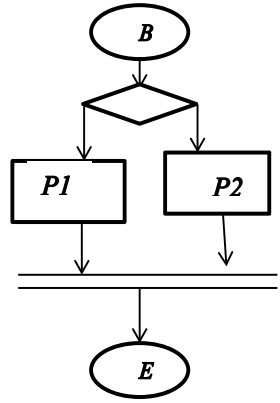


Рис. 21

Показано, что алгоритм корректен, если в нем отсутствуют фрагменты указанных типов. Сложность состоит в том, что в этих фрагментах под операторными вершинами *P*1 и *P*2 понимаются различные подграфы, что требует при проверке корректности анализа всей ПГСА. Это делает задачу *NP*-полной.

6.2.3. Построение корректных ПГСА

Метод основан на том, что выделяется несколько базовых фрагментов (структурированный базис), на основе которых строится ПГСА. ПГСА, построенные в этом базисе, являются корректными и называются структурированными. Базовыми структурами являются:

1) [*P*] — описание программы (рис. 22).

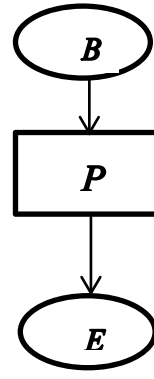


Рис. 22

2) Конкатенация (рис. 23).

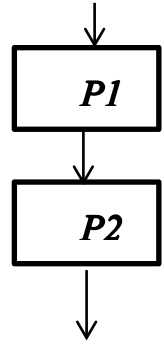


Рис. 23

3) Итерации (рис. 24).

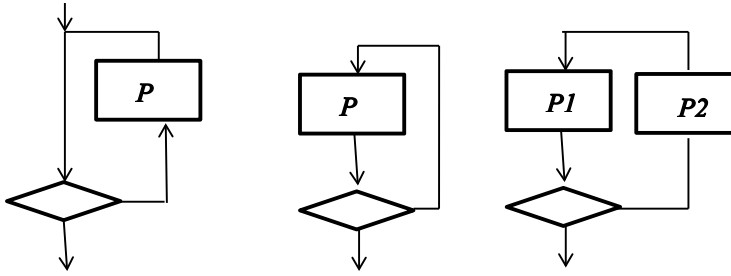


Рис. 24

4) Альтернатива (рис. 25).

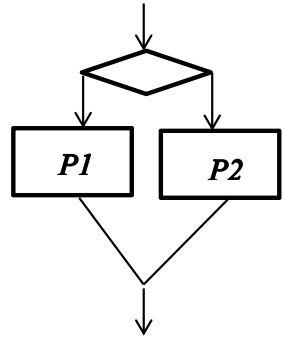


Рис. 25

5) Тасование (рис. 26).

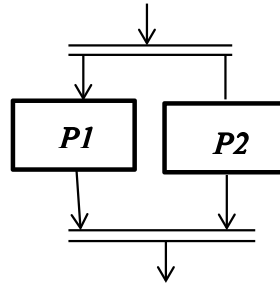


Рис. 26

Пример ПГСА в структурированном виде (рис. 27).

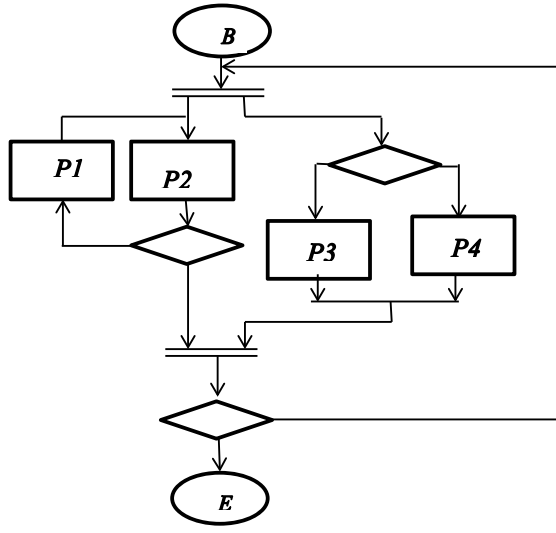


Рис. 27

Структурированные ПГСА всегда являются корректными, обратное в общем случае неверно, существуют корректные, но не структурированные ПГСА.

Один из способов определения корректности ПГСА — после его построения перейти к другой модели, на другом языке. И по ней уже сделать вывод о корректности. Пример такой модели — сети Петри.

6.3. Сети Петри для проверки корректности

6.3.1. Основные понятия

Сеть Петри $N = <P, T, X, M_0>$, где

- $P = \{P_i \mid i = 1, ...\}$ — множество позиций (мест);
- $T = \{t_i \mid i = 1, ...\}$ — множество переходов;
- $X \subseteq (P \times T) \times (T \times P)$ — бинарное отношение на $\{P \cup T\}$, определяет дуги ориентированного графа;
- M_0 — отображение $P \rightarrow \{0, 1, 2, 3...\}$, определяет начальное маркирование.

Пример графического изображения сети Петри приведен на рис. 28.

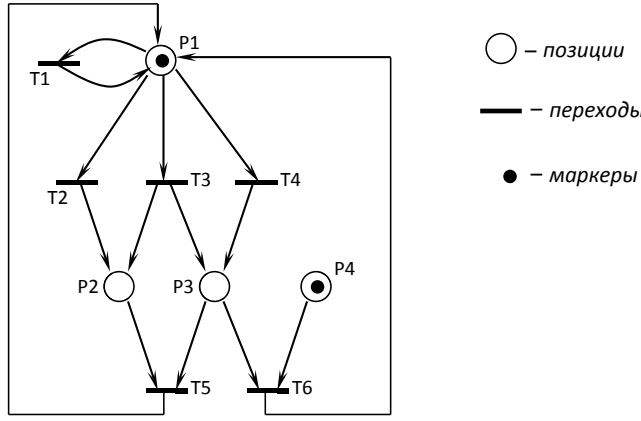


Рис. 28

M_0: P_1–1, P_2–0, P_3–0, P_4–1.

Будем обозначать M_0 в виде вектора (1001), где число определяет количество маркеров позиции:

A $\subset \{P \cup T\}$; $^{\cdot}A$ — входы в A; $A^{\cdot}$ — выходы A.

Переход возбужден, если все позиции, связанные с ним по заходящим дугам (входные позиции перехода), имеют ненулевое число маркеров.

Один из возбужденных переходов может сработать, при этом число маркеров во всех входных позициях перехода уменьшится на единицу, а во всех выходных позициях перехода увеличится на единицу.

Например, если сработает переход Т 3 в примере, приведенном на рис. 3, то маркировка изменится так: 1001 → 0111 (P1→T3→P2, P3).

6.3.2. Способы описания сетей Петри

· Матрица входов и выходов

Сеть Петри, как и любой граф, допускает способ описания в виде матрицы смежности. Так как граф является двудольным, то для его описания берут две части из матрицы, как показано на рис. 29.

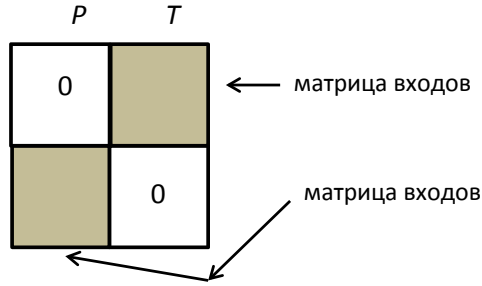


Рис. 29

· Граф достижимых маркирований

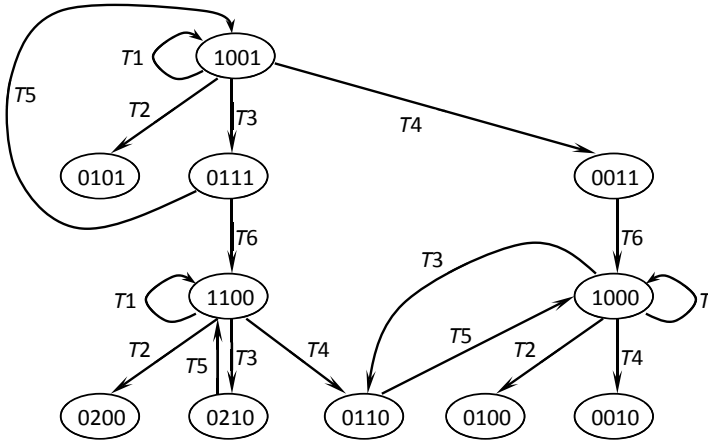


Рис. 30

Если в какой-либо вершине сети Петри возможно накопление маркеров, то число возможных маркирований графа будет бесконечно большим, что приведет к графу бесконечных размеров. Чтобы возможно было построить граф в этом случае, в таких ситуациях вместо числа накопленных маркеров ставят символ «ω», в результате чего получается конечный рисунок.

P_1	→	P_1
P_1	→	P_2
P_1	→	$P_2 P_3$
P_1	→	P_3
$P_2 P_3$	→	P_1
$P_3 P_4$	→	P_1

Выражение срабатывания (абстрактная микропрограмма) описывает перемещение меток при срабатывании всех переходов.

6.3.3. Правильные сети Петри

Маркирование M' достижимо из M, если найдется путь $\sigma = T_{i1}, T_{i2} \ldots T_{ik}$ (цепь срабатываний) в графе достижимости из М в M». Обозначается как $M \rightarrow M'$.

Переход Т псевдоживой при маркировании M, если $\exists M'$: $M \rightarrow M' \&$ $T \in \sigma$ (то есть переход при данном маркировании может сработать).

Переход Т живой, если для любого M': М → M' найдется σ: М → M'& (Т принадлежит σ).

Позиция P при маркировании M k-ограниченная, если для любого M': $M \rightarrow M'$ и число меток в позиции Р: $M(P) \leqslant k$.

Одно-ограниченная позиция называется безопасной.

Сеть живая при маркировании M, если все ее переходы живые.

Сеть безопасная при маркировании M, если все ее переходы безопасные.

Если сеть при маркировании M живая и безопасная, то маркирование правильное.

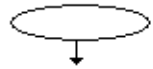
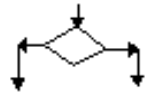
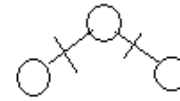
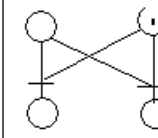
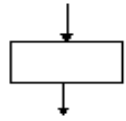
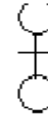
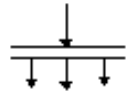
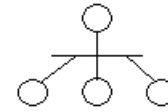
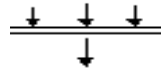
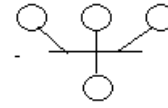
Сеть правильная, если для нее существует правильное маркирование.

Живые сети обозначаются L, неживые — $\bar{L}$, безопасные — S, небезопасные — $\bar{S}$. По этим признакам получаем четыре класса сетей: LS и $L\bar{S}$, $\bar{L}S$ и $\bar{L}\bar{S}$.

6.4. Связь ПГСА с сетью Петри

Переход от ПГСА к сети Петри производится заменой вершин ПГСА на фрагменты сети Петри в соответствии со следующими правилами сопоставлений (табл. 20).

Таблица 20

Имя вершины ПГСА	Вид в ПГСА	Представление в сети Петри	Другое представление
Begin			—
End			—
If			
Process			—
Fork			—
Join			—

Объединение фрагментов связано с отождествлением выходных позиций одного фрагмента с входными позициями второго. Естественно, что число вершин в сети Петри будет больше числа вершин в исходной ПГСА, что связано с тем, что выходная модель проще по числу типов вершин. ПГСА корректна, если сопоставленная таким образом сеть Петри правильная.

РЕКОМЕНДОВАННАЯ ЛИТЕРАТУРА

1. Ожиганов А. А. Теория автоматов : учебное пособие. СПб. : НИУ ИТМО, 2013. 84с.

2. Норенков И. П. Основы автоматизированного проектирования : учеб. для вузов. 4-е изд., перераб. и доп. М. : Изд-во МГТУ им. Н. Э. Баумана, 2009. 430 с.

3. Боровков А. И. [и др]. Компьютерный инжиниринг. Аналитический обзор: учебное пособие. СПб. : Изд-во Политехн. ун-та, 2012. 93 с.

4. Малышкин В. Э. Основы параллельных вычислений [Электронный ресурс]. URL: http://www.ssga.rU/metodich/Paral1/contents.html (дата обращения: 14.12.2014).

ОГЛАВЛЕНИЕ